(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 304 540 B1**

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication of patent specification: **20.04.94**

(51) Int. Cl.5: **G06F 9/44**, G06F 11/00, G06F 15/16, G06F 13/12

(21) Application number: **88105007.4**

(22) Date of filing: **05.10.82**

(60) Publication number of the earlier application in accordance with Art.76 EPC: **0 077 008**

(54) A method of initializing a data processing system.

(30) Priority: **05.10.81 US 308826**

(43) Date of publication of application:
01.03.89 Bulletin 89/09

(45) Publication of the grant of the patent:
20.04.94 Bulletin 94/16

(84) Designated Contracting States:
DE FR GB

(56) References cited:
EP-A- 0 012 018
EP-A- 0 031 484

IBM TECHNICAL DISCLOSURE BULLETIN, vol. 14, no. 1, June 1971, pages 44,45, New York, US; G.H. EDSTROM et al.: "Subsystem initialization"

IBM TECHNICAL DISCLOSURE BULLETIN, vol. 22, no. 12, May 1980, page 5255, New York, US; J.R. VOLK et al.: "Sense state command"

(73) Proprietor: **DIGITAL EQUIPMENT CORPORATION**
111 Powdermill Road
Maynard Massachusetts 01754-1418(US)

(72) Inventor: **Rubinson, Barry L.**
585 Anaconda Drive
Colorado Springs Colorado 80919(US)
Inventor: **Gardner, Edward A.**
1262 Hofstead Terrace
Colorado Springs Colorado 80907(US)
Inventor: **Grace, William A.**
535 Silver Springs Circle
Colorado Springs Colorado 80919(US)
Inventor: **Lary, Richard F.**
1140 Big Valley Drive
Colorado Springs Colorado 80919(US)
Inventor: **Keck, Dale R.**
833 Hans Brinker Street
Colorado Springs Colorado 80907(US)

(74) Representative: **Betten, Jürgen, Dipl.-Ing. et al**
Patentanwälte Betten & Resch
Reichenbachstrasse 19
D-80469 München (DE)

Rank Xerox (UK) Business Services
(3.10/3.09/3.3.3)

Description

Field of the Invention

This invention relates to the field of data processing systems and, in particular to a method of initializing the data processing system.

Background of the Invention

In data processing systems utilizing secondary storage facilities, communications between the host processor, or main frame, and secondary storage facilities have a considerable impact on system performance. Secondary storage facilities comprise data processing elements which are not integral parts of a central processing unit (CPU) and its random access memory element (i.e., the host computer), but which are directly connected to and controlled by the central processing unit or other elements in the system. These facilities are also known as "mass storage" elements or subsystems and include, among other possibilities, disk-type or tape-type memory unite (also called drives).

A secondary storage facility typically includes a controller and one or more drives connected thereto. The controller appears to the rest of the system as simply an element on an input-output bus which connects together the various elements of the system. It receives commands and provides responses and other messages over the bus. These commands include information about operations to be performed by the storage facility, including, for example, the drive to be used, the size of the transfer and perhaps the starting address on the drive for the transfer and the starting address on some other system element, such as the random access memory unit of the host, to or from which the data is to be transferred. The controller converts this command information into the necessary signals to effect the transfer between the appropriate drive and other system elements. During the transfer itself, the controller routes the data between the drive and the input/output bus or a memory bus.

Typically, controllers have communicated with a host CPU at least partially by means of an interrupt mechanism. That is, when one of a predetermined number of significant events have occurred, the controller has generated an interrupt request signal which the host has received a short time later; in response, the host has stopped what it was doing and has conducted a dialogue with the controller to service the controller's operation. Consequently, every interrupt request signal generated by the controller gave rise to a delay in the operation of the central processor.

Modern controllers for secondary storage facilities are usually so-called "intelligent" devices, containing one or more processors of their own, allowing them to perform sophisticated tasks with some degree of independence. Sometimes, a processor in a controller will share a resource with another processor, such as the host's central processor unit. One resource which may be shared is a memory unit.

It is well known that when two independent processors share a common resource (such as a memory through which the processors and the processes they execute may communicate with each other), the operation of the two processors (i.e., the execution of processes or tasks by them) must be "interlocked" or "synchronized", so that in accessing the shared resource, a defined sequence of operation is maintained and so-called "race" conditions are avoided. That is, once a first processor starts using a shared resource, no other processor may be allowed to access that resource until the first processor has finished operating upon it. Operations which otherwise might have occurred concurrently must be constrained to take place serially. Otherwise, information may be lost, a processor may act on erroneous information, and system operation will be unreliable. To prevent this from happening, the communications apparatus (i.e., bus) which links together the processors and the shared resource typically is provided with a hardware "interlock" or synchronization capability, by means of which each processor is prevented from operating on the shared resource in other than a predefined sequence.

An illustration may help to demonstrate the problem. Consider two processors trying to communicate with each other through a memory which serves as a shared resource: a first one of the processors is supposed to perform a calculation and put the result X in memory location Y, following which the other (i.e., second) processor is supposed to take the result in location Y and use it in a calculation to produce result Z. If the second processor reads the contents of location Y before the first processor has written X there, result Z will be wrong. Similarly, if the first processor not only writes X into location Y, but before the second processor gets a chance to read X it overwrites into location Y some new data X', the second processor once again will not receive the correct information if it thereafter reads location Y. This loss of proper sequencing, which allows one processor to "get ahead" of the other and lose context, is referred to as a race condition.

In the prior art, three interlock mechanisms are widely known for synchronizing processes within an operating system, to avoid race conditions. One author calls these mechanisms (1) the test-and-set

instruction mechanism, (2) the wait and signal mechanism and (3) the P and V operations mechanism. S. Madnick and J. Donovan, Operating Systems, Section 4-5.2 at 251-55 (McGraw-Hill, Inc., 1974). That text is hereby incorporated by reference for a description and discussion of those mechanisms. Another author refers to three techniques for ensuring correct syunchronization when multiple processors communicate through a shared memory as (1) process synchronization by semaphores, (2) process synchronization by monitors and (3) process synchronization by monitors without mutual exclusion. C. Weitzman, Distributed Micro/Minicomputer Systems: Structure, Implementation and Application, Section 3.2 at 103-114 (Prentice-Hall, Inc., 1980). That text is hereby incorporated by reference for a description and discussion of those techniques. When applied to multiple processors which communicate with a shared resource via a bus, such mechanisms impose limitations on bus characteristics; they require, for example, that certain compound bus operations be indivisible, such as an operation which can both test and set a so-called "semaphore" or monitor without being interrupted while doing so. These become part of the bus description and specifications.

If the testing of a semaphore were done during one bus cycle and the setting during a different bus cycle, two or more processors which want to use a shared resource might test its semaphore at nearly the same time. If the semaphore is not set, the processors all will see the shared resource as available. They then will try to access it; but only one can succeed in setting the semaphore and gaining access; each of the other processors, though, having already tested and found the resource available, would go through the motions of setting the semaphore and reading or writing data without knowing it had not succeeded in accessing the resource.The data thus read would be erroneous and the data thus written could be lost.

Not all buses, though, are designed to allow implementation of such indivisible operations, since some buses were not designed with the idea of connecting multiple processors via shared resources. Consequently, such buses are not or have not been provided with hardware interlock (or processor synchronization) mechanisms. One bus in this category is the UNIBUS of Digital Equipment Corporation, assignee hereof.

When a bus does not have such a capability, resort frequently has been made to using processor interrupts to control the secondary storage facility, or some combination of semaphores and interrupts (as in the Carnegie-Mellon University C.mmp multiminicomputer system described at pages 27-29 and 110-111 of the above-identified book by Weitzman), but those approaches have their drawbacks. If multiple processors on such a hug operate at different rates and have different operations to perform, at least one processor frequently may have to wait for the other. This aggravates the slow-down in processing already inherent in the use of interrupt control with a single processor.

A further characteristic of prior secondary storage facilities is that when a host computer initially connects to a controller, it usually assumes, but cannot then verify, that the controller is operating correctly.

EP-A-0031484 (IBM) relates to a data transmission technique within a distributed data processing system including one or more host computers, a storage subsystem adapter, and secondary storage devices. Communication within this system is determined by the information contained in the various message formats contemplated by the reference. The system described in this reference uses a configuration element to forward messages received by the host computer to the ATTENTION of host system software (page 46, section 6). The configuration element contains state information which is accessed and used by the host to facilitate the subsystem communications and this state information may be derived through initialization procedures or altered by host computer software (page 47, paragraph 1).

IBM TECHNICAL DISCLOSURE BULLETIN, Vol. 14, No. 1, June 1971; G.H. EDSTROM et al. "Subsystem Initialization" relates to a "hard wire" initialization technique which is described as useful to ensure the integrity of the I/O channel between a CPU and a control unit. This technique includes the use of logic gates and programmed diagnostic routines such as forcing logic states of particular components to all "ones" or all "zeros" to determine malfunctions. This reference also describes the use of "hang" conditions which require operator intervention when malfunctions are detected.

It is an object of this invention to provide a method of initializing a data processing system by a communications apparatus operates between a controller and a host for permitting the host to verify correct operation of the controller during initialization.

Summary of the Invention

According to this invention as claimed there is provided a method of initializing a data processing system which includes a host processor, mass storage means, mass storage control means for controlling the operation of the mass storage means, and bus means for interconnecting the host processor and the mass storage control means to

enable communication therebetween, a register connected to the bus means and through which various information may be transmitted between the host processor and the mass storage control means, and said register having multiple locations for storing bit values.

Brief Description of the Drawings

Figure 1 is a conceptual block diagram of a system in which the initializing method of the present invention has utility;
Figure 2 is a basic block diagram of a data processing system in which the method of the present invention may be employed;
Figure 3A is a system block diagram of an illustrative embodiment of a data processing system utilizing the initializing method of the present invention;
Figures 3B and 3C are diagrammatic illustrations of a ring 80D or 80E of Figure 3A;
Figures 4A and 4B are elementary flow diagrams illustrating the sequence of events that occurs when the controller wishes to send a response to the host;
Figure 5 is an elementary flow diagram showing the sequence of events that occurs when the host issues a command to the controller;
Figure 6 is a similar flow diagram showing the controller's action in response to the host's issurance of a command;
Figure 7 is a diagrammatic illustration of the communications areas of host memory, including the command and response rings;
Figure 8 is a diagrammatic illustration of the formatted command and response descriptors which comprise the ring entries;
Figure 9 is a diagrammatic illustration of the command and response message envelopes;
Figure 10 is a diagrammatic illustration of a buffer descriptor;
Figure 11 is a diagrammatic illustration of the status and address (SA) register 38 of Figure 3A;
Figures 12A-12D are flow charts of the port/controller initialization sequence according to this invention; and
Figure 13 is a diagrammatic illustration of the "last fail" response packet used in this invention.

Detailed Description of an Illustrative Embodiment

The present invention has particular utility in a data processing system having an architectural configuration designed to enhance development of future mass storage systems, at reduced cost. Such a system is shown in Figure 1. In this system, a high level input/output (indicated at 1) protocol is employed for communications between a host computer 2A and an intelligent mass storage controller 2B. Such a high level protocol is intended to free the host from having to deal with peripheral device-dependent requirements (such as disk geometry and error recovery strategies). This is accomplished in part through the use of a communications hierachy in which the host communicates via only one or two peripheral device "class" drivers, such as driver 4, instead of a different I/O driver for each model of peripheral device. For example, there may be one driver for all disk class devices and another for all tape class devices.

Device classes are determined by their storage and transfer characteristics. For example a so-called "disk class" is characterized by a fixed block length, individual block update capability, and random access. Similarly a so-called "tape class" is characterized by a variable block length, lack of block update capability, and sequential access. Thus, the terms "disk" and "tape" as used herein refer to devices with such characteristics, rather than to the physical form of the storage medium.

Each class driver, in turn, communicates with a device controller (e.g., 2B) through an interface mechanism 10.

Much of the interface apparatus 10 is bus- (rather than drive-) specific. Therefore, when it is desired to connect a new mass storage device to the system, there is no need to change the host's input/output processes or operating system, which are costly (in time, as well as money) to develop. Only the controller need be modified to any substantial degree, which is far less expensive. And much of that cost can be averted if the controller and host are made self-adaptive to certain of the storage device's characteristics, as explained in the above-identified commonly assigned applications.

Within the framework of this discussion, a data processing system comprises a plurality of subsystems interconnected by a communications mechanism (i.e., a bus and associated hardware). Each subsystem contains a port driver, which interfaces the subsystem to the communications apparatus. The communications apparatus contains a port for each subsystem; the port is simply that portion of the communication apparatus to which a port driver interfaces directly.

Thus, in Fig. 1 the exemplary system comprises host computer 2A, intelligent mass storage controller 2B and communications apparatus 7. Host 2A includes a peripheral class driver 3 and a port driver 4. Controller 2B, in turn, includes a counterpart port driver 5 and an associated high-level protocol server 6. The communications apparatus 7 includes an interface member (8, 9) for

each port driver.

The port drivers 4 and 5 provide a standard set of communications services to the processes within their subsystems; port drivers cooperate with each other and with the communications apparatus to provide these services. In addition, the port drivers shield the physical characteristics of the communications apparatus from processes that use the communications services.

Class driver 3 is a process which executes within host 1. Typically, a host I/O class driver 3 communicates with a counterpart in the controller 2B, called a high-level protocol server.

The high-level protocol server processes host commands, passes commands to device-specific modules within the controller, and sends responses to host commands back to the issuing class driver.

In actual implementation, it is also possible for the functions of the controller-side port driver 5 and interface 9 to be performed physically at the host side of the communications apparatus 7. This is shown in the example described below. Nevertheless, the diagram of Fig. 1 still explains the architectural concepts involved.

Note also that for purposes of the further explanation which follows, it is generally unnecessary to distinguish between the interface and its port driver. Therefore, unless the context indicates otherwise, when the word "port" is used below, it presumes and refers to the inclusion of a port driver with an interface.

Referring now to Fig. 2, there is shown a system level block diagram of a data processing system utilizing the present invention. A host computer 2A (including an interface apparatus 10) employs a secondary storage subsystem 20 comprising a controller 30, a disk drive 40 and a controller-drive interconnection cable 50. The host communicates with the secondary storage subsystem over an input/output bus 60.

Fig. 3A expands the system definition to further explain the structure of the host 2A, controller 30 and their interconnection via interface apparatus 10. As illustrated there, the host 2A comprises four primary subunits; a central procesor unit (CPU) 70, a main memory 80, a system bus 90 and a bus adapter 110.

A portion 80A of memory 80 is dedicated to service as a communications region for accessing the remainder of memory 80. As shown in Fig. 3A, communications region 80A comprises four subregions, or areas. Areas 80B and 80C together form the above-indicated header section of the communications region. Area 80B is used for implementing a bus adapter purge function and area 80C holds the ring transition interrupt indicators used by the port. The variable-length section of the communications region comprises the response list area 80D and the command list area 80E. The lists in areas 80D and 80E are organized into rings. Each entry, in each ring, in turn, contains a descriptor (see Fig. 10) pointing to a memory area of sufficient size to accommodate a command or response message packet of predetermined maximum length, in bytes.

Host 2A may, for example, be a Model VAX-11/780 or PDP 11 computer system, marketed by Digital Equipment Corporation of Maynard, Massachusetts.

System bus 90 is a bi-directional information path and communications protocol for data exchange between the CPU 70, memory 80 and other host elements which are not shown (so as not to detract from the clarity of this explanation). The system bus provides checked PARALLEL information exchanges synchronous with a common system clock. A bus adapter 110 translates and transfers signals between the system bus 90 and the host's input/output (I/O) bus 60. For example, the I/O bus 60 may be the UNIBUS I/O connection, the system bus may be the synchronous backplane interconnection (SBI) of the VAX-11/780 computer, and the bus adapter 110 may be the Model DW780 UNIBUS Adapter, all Digital Equipment Corporation products.

Controller 2B includes several elements which are used specifically for communicating with the host 2A. There are pointers 32 and 34, a command buffer 36 and a pair of registers, 37 and 38. Pointers 32 and 34 keep track of the current host command ring entry and the host response ring entry, respectively. Command buffers 36 provide temporary storage for commands awaiting processing by the controller. Register 37, termed the "IP" register, is used for initialization and polling. Register 38, termed the "SA" register, is used for storing status and address information. A processor 31 is the "heart" of the controller 2B; it executes commands from buffers 36 and docs all the housekeeping to keep communication flowing between the host 2A and the drive 40.

The physical realization of the transport mechanism includes the UNIBUS interconnection (or a suitable counterpart) 60, system bus 90 and any associated host and/or controller-based logic for adapting to same, and bus adapter 110.

The operation of the rings may be better understood by referring to Figs. 3B and 3C, where an exemplary four-entry ring 130 is depicted. This depiction is conceptual, rather than physical. The ring 130 may be either a command ring or a response ring, since only their application differs. There are four ring entry positions 132, 134, 136 and 138 consecutive addresses RB, RB+4, respectively. Each ring entry has associated with it an ownership bit (133, 135, 137, 139) which is used

to indicate its status. Assume the ring 130 has been operating for some time and we have started to observe it at an arbitrarily selected moment, indicated in Fig. 3B. A write pointer (WP), 142, points to the most recent write entry; correspondingly, a read pointer (RP), 144, points to the most recent read entry. In Fig. 3B, it will be seen that entry 138 has been read, as indicated by the position of RP 144 and the state of ownership bit 139. By convention, the ownership bit is set to 1 when a location has been filled (i.e., written) and to 0 when it has been emptied (i.e., read). The next entry to be read is 132. Its ownership bit 133 is set to 1, indicating that it already has been written. Once entry 132 is read, its ownership bit is cleared, to 0, as indicated in Fig. 3C. This completely empties the ring 130. The next entry 134 cannot be read until it is written and the state of ownership bit 135 is changed. Nor can entry 132 be re-read accidentally, since its ownership bit has been cleared, indicating that it already has been read.

Having thus provided a block diagram exmplanation of the invention, further understanding of this interface will require a brief digression to explain packet communications over the system.

The port is a communications apparatus in which communications take place between pairs of processors resident in separate subsystems. (As used herein, the term "subsystems" include the host computers and device controllers; the corresponding processes are host-resident class drivers and controller-resident protocol servers.)

Communications between the pair of processes take place over a "connection" which is a soft communications path through the port; a single port typically will implement several connections concurrently. Once a connection has been established, the following three services are available across that connection: (1) sequential message; (2) datagram and (3) block data transfer.

When a connection is terminated, all outstanding communications on that connnection are discarded; that is, the receiver "throws away" all unacknowledged messages and the sender "forgets" that such messages have been sent.

The implementation of this communications scheme on the UNIBUS interconnection 60 has the following characteristics: (1) communications are always point-to-point between exactly two subsystems, one of which is always the host; (2) the port need not be aware of mapping or memory management, since buffers are identified with a UNIBUS address and are contiguous within the virtual bus address space; and (3) the host need never directly initiate a block data transfer.

The port effectively and conceptually is integral with the controller, even though not physically localized there. This result happens by virtue of the point-to-point property and the fact that the device controller knows the class of device (e.g., disk drive) which it controls; all necessary connections, therefore, can be established by the port/controller when it is initialized.

The Sequential Message service guarantees that all messages sent over a given connection are transmitted sequentially in the order originated, duplicate-free, and that they are delivered. That is, messages are received by the receiving process in the exact order in which the sending process queued for the transmission. If these guaratees cease to be met, or if a MESSAGE cannot be delivered for any reason, the port enters the so-called "fatal error" state (described below) and all port connections are terminated.

The Datagram services does not guarantee reception, sequential reception of duplicate-free reception of datagrams, though the probability of failure may be required to be very low. The port itself can never be the cause of such failures; thus, if the using processes do make such guarantees for datagrams, then the datagram service over the port becomes equivalent to the Sequential Message service.

The Block Data Transfer service is used to move data between named buffers in host memory and a peripheral device controller. In order to allow the port to be unaware of mapping or memory management, the "name" of a buffer is merely the bus address of the first byte of the buffer. Since the host never directly initiates a block data transfer, there is no need for the host to be aware of controller buffering.

Since the communicating processes are asynchronous, flow control is needed if a sending process is to be prevented from producing congestion or deadlock in a receiving process (i.e., by sending messages more quickly than the receiver can capture them). Flow control simply guarantees that the receiving process has buffers in which to place incoming messages; if all such buffers are full, the sending process is forced to defer transmission until the condition changes. Datagram service does not use flow control. Consequently, if the receiving process does not have an available buffer, the datagram is either processed immediately or discarded, which possibility explicitly is permitted by the rules of that service. By contrast, the Sequential Message services does use flow control. Each potential receiving process reserves, or pre-allocates, some number of buffers into which messages may be received over its connection. This number is therefore the maximum number of messages which the sender may have outstanding and unprocessed at the receiver, and it is communicated to the sender by the receiver in the form of a "credit" for the connection. When a sender has

used up its available credit, it must wait for the receiver to empty and make available one of its buffers. The message credits machinery for the port of the present invention is described in detail below.

The host-resident driver and the controller provide transport mechanism control facilities for dealing with: (1) transmission of commands and responses: (2) sequential delivery of commands; (3) asynchronous communication; (4) unsolicited responses; (5) full duplex communications; and (6) port failure recovery. That is, commands, their responses and unsolicited "responses" (i.e., controller-to-host messages) which are not responsive to a command may occur at any time; full duplex communication is necessary to handle the bi-directional flow without introducing the delays and further buffering needs which would be associated with simplex communications. It is axiomatic that the host issues commands in some sequence. They must be fetched by the controller in the order in which they were queued to the transport mechanism, even if not executed in that sequence. Responses, however, do not necessarily occur in the same order as the initiating commands; and unsolicited messages can occur at any time. Therefore, asynchronous communications are used in order to allow a response or controller-to-host message to be sent whenever it is ready. Finally, as to port failure recovery, the host's port driver places a timer on the port, and reinitializes the port in the event the port times out.

This machinery must allow repeated access to the same host memory location, whether for reads, writes, or any mixture of the two.

The SA and IP registers (37 and 38) are in the I/O page of the host address space, but in controller hardware. They are used for controlling a number of facets of port operation. These registers are always read as words. The register pair begins on a longword boundary. Both have predefined addresses. The IP register has two functions: first, when written with any value, it causes a "hard" initialization of the port and the device controller; second, when read while the port is operating, it causes the controller to initiate polling of the command ring, as discussed below. The SA register 38 has four functions; first, when read by the host during initialization, it communicates data and error information relating to the initialization process; second, when written by the host during initialization, it communicates certain host-specific parameters to the port; third, when read by the host during normal operation, it communicates status information including port - and controller-detected fatal errors; and fourth, when zeroed by the host during initialization and normal operation, it signals the port that the host has successfully completed a bus adapter purge in response to a port-initiated purge request.

The port driver in the host's operating system examines the SA register regularly to verify normal port/controller operation. A self-detected port/controller fatal error is reported in the SA register as discussed below.

Transmission of Commands and Responses - Overview

When the controller desires to send a response to the host, a several step operational sequence takes place. This sequence is illustrated in Figs. 4A and 4B. Initially, the controller looks at the current entry in the response ring indicated by the response ring pointer 34 and determines whether that entry is available to it (by using the "ownership" bit). (Step 202.) If not, the controller continues to monitor the status of the current entry until it becomes available. Once the controller has access to the current ring entry, it writes the response into a response buffer in host memory, pointed to by that ring entry, and indicates that the host now "owns" that ring entry by clearing an "Ownership" bit; it also sets a "FLAG" bit, the function of which is discussed below. (Step 204.)

Next, the port determines whether the ring has gone from an empty to a non-empty transition (step 206); if so, a potentially interruptable condition has occurred. Before an interrupt request is generated, however, the port checks to ensure that the "FLAG" bit is a 1 (step 208); an interrupt request is signalled only on an affirmative indication (Step 210).

Upon receipt of the interrupt request, the host, when it is able to service the interrupt, looks at the current entry in the response ring and determines whether it is "owned" by the host or controller (ie.e., whether it has yet been read by that host). (Step 212.) If it is owned by the controller, the interrupt request is dismissed as spurious. Otherwise, the interrupt request is treated as valid, so the host PROCESSES the response (Step 214) and then updates its ring pointer (Step 216).

Similar actions take place when the host wants to send a command, as indicated in Fig. 5. To start the sequence, the host looks at the current command ring entry and determines whether that ring is owned by the host or controller. (Step 218) If it is owned by the controller, the host starts a timer (Step 220.) (provided that is the first time it is looking at that ring entry; if the timer is not stopped (by the command ring entry becoming available to the host) and is allowed to time out, a failure is indicated; the port is then reinitialized. (Step 222.) If the host owns the ring entry, however, it puts the packet address of the command in the current ring

entry. (Step 224.) If a command ring transfer interrupt is desired (Step 226), the FLAG bit is set = 1 to so indicate (Step 228). The host then sets the "ownership" bit = 1 for the ring entry to indicate that there is a command in that ring (i.e., the host reads the IP register, which action is interpreted by the port as a notification that the ring contains one or more commands awaiting transmission); in response, the port steps through the ring entries one by one until all entries awaiting transmission have been sent. (Step 232.)

The host next determines whether it has additional commands to send. (Step 234) If so, the process is repeated; otherwise, it is terminated.

In responding to the issuance of a command (see Fig. 6), the port first detects the instruction to poll (i.e., the read operation to the IP register). (Step 234) Upon detecting that signal, the port must determine whether there is a buffer available to receive a command. (Step 236) It waits until the buffer is available and then reads the current ring entry to determine whether that ring entry is owned by the port or host. (Step 238) If owned by the port, the command packet is read into a buffer. (Step 240) The FLAG bit is then set and the "ownership" bit in the ring entry is changed to indicate host ownership (Step 242.) If not owned by the port, polling terminates.

A test is then performed for interrupt generation. First the port determines whether the command ring has undergone a full to not-full transition. (Step 244) If so, the port next determines whether the host had the FLAG bit set. (Step 246.) If the FLAG bit was set, an interrupt request is generated. (Step 248.) In either case, the ring pointer is then incremented. (Step 250).

Response packets continue to be removed after the one causing an interrupt and, likewise, command packets continue to be removed by the port after poll.

The Communications Area

The communications area is aligned on a 16-bit word boundary whose layout is shown in Fig. 7. Addresses for the words of the rings are identified relative to a "ringbase" address 252. The words in regions 80B, 80C whose addresses are ringbase-3, ringbase-2 and ringbase-1 (hereinafter designated by the shorthand [ringbase-3], etc., where the brackets should be read as the location "whose address is") are used as indicators which are set to zero by the host and which are set non-zero by the port when the port interrupts the host, to indicate the reason for the interrupt. Word [ringbase-3] indicates whether the port is requesting a bus adapter purge; the non-zero value is the adapter channel number contained in the high-order byte 254 and derived from the triggering command. (The host responds by performing the purge. Purge completion is signalled by writing zeros to the SA register).

Word 256 [ringbase-2] signals that the command queue has transitioned from full to not-full. Its non-zero value is predetermined, such as one. Similarly, word 258 [ringbase-1] indicates that the response queue has transitioned from empty to not-empty. Its non-zero value also is predetermined (e.g., one).

Each of the command and response lists is organized into a ring whose entries are 32-bit descriptors. Therefore, for each list, after the last location in the list has been addressed, the next location in sequence to be addressed is the first location in the list. That is, each list may be addressed by modulo-N counter, where N is the number of entries in the ring. The length of each ring is determined by the relative speeds with which the host and the port/controller generate and process messages; it is unrelated to the controller command limit. At initialization time, the host sets the ring lengths.

Each ring entry, or formatted descriptor, has the layout indicated in Fig. 8. In the low-order 16-bits (260), the least significant bit, 262, is zero; that is, the envelope address [text + 0] is word-aligned. The remaining low-order bits are unspecified and vary with the data. In the high-order portion 264 of the descriptor, the letter "U" in bits 266 and 268 represent a bit in the high-order portion of an 18-bit UNIBUS (or other bus) address. Bits 270-276, labelled "Q", are abvailable for extending the high-order bus address; they are zero for UNIBUS systems. The most significant bit, 278, contains the "ownership" bit ("O") referred to above; it indicates whether the descriptor is owned by the host (0 = 1), and acts as an interlock protecting the descriptor against premature access by either the host or the port. The next lower bit, 280, is a "FLAG" bit (labelled "F") whose meaning varies depending on the state of the descriptor. When the port returns a descriptor to the host, it sets F = 1, indicating that the descriptor is full and points to response. On the other hand, when the controller acquires a descriptor from the host, F = 1 indicates that the host wants a ring transition interrupt due to this slot. It assumes that transition interrupts were enabled during initialization and that this particular slot triggers the ring transition. F = 0 means that the host does not want a transition host interrupt, even if interrupts were enabled during initialization. The port always sets F = 1 when returning a descriptor to the host; therefore, a host desiring to override right transition interrupts must always clear the FLAG bit when passing ownership of a descriptor to the port.

Message Envelopes

As stated above, messages are sent as packets, with an envelope address pointing to word [text + 0] of a 16-bit, word-aligned message envelope formatted as shown in Fig. 9.

The MSG LENGTH field 282 indicates the length of the message text, in bytes. For commands, the length equals the size of the command, starting with [text __ 0]. For responses, the host sets the length equal to the size of the response buffer, in bytes, starting with [text + 0]. By design, the minimum acceptable size is 60 bytes of message text (i.e., 64 bytes overall).

The message length field 282 is read by the port before the actual transmission of a response. The port may wish to send a response longer than the host can accept, as indicated by the message length field. In that event, it will have to break up the message into a plurality of packets of acceptable size. Therefore, having read the message length field, the controller then sends a response whose length is either the host-specified message length or the length of the controller's response, if smaller. The resulting value is set into the message length field and sent to the host with the message packet. Therefore, the host must re-initialize the value of tha field for each proposed reponse.

The message text is contained in bytes 284a-284m, labelled MBj. The "connection id" field 286 identifies the connection serving as source of, or destination for, the message in question. The "credits" field 288 gives the credit value associated with the message, which is discussed more fully below. The "msgtyp" field 290 indicates the message type. For example, a zero may be used to indicate a sequential message, wherein the credits and message length fields are valid. A one may indicate a datagram, wherein the credits field must be zero, but message length is valid. Similarly, a two may indicate a credit notification, with the credits field valid and the message length field zero.

Message Credits

A credit-based message limit apparatus is employed for command and response flow control. The credits field 288 of the message envelope supports the credit-accounting algorithm. The controller 30 has a buffer 36 for holding up to M commands awaiting execution. In its first response, the controller will return in the credits field the number, M, of commands its buffer can hold. This number is one more than the controller's acceptance limit for non-immediate commands; the "extra" slot is provided to allow tht host always to be able to issue an immediate-class command. If the credit account has a value of one, then the class driver may issue only an immediate-type command. If the account balance is zero, the class driver may not issue any commands at all.

The class driver remembers the number M in its "credit account". Each time the class driver queues a command, it decrements the credit account balance by one. Conversely, each time the class driver receives a response, it increments the credit account balance by the value contained in the credits field of that response. For unsolicited responses, this value will be zero, since no command was executed to evoke the response; for solicited responses, it normally will be one, since one command generally gives rise to one response.

For a controller having M greater than 15, responses beyond the first will have credits greater than one, allowing the controller to "walk" the class driver's credit balance up to the correct value. For a well-behaved class driver, enlarging the command ring beyond the value M+1 provides no performance benefits; in this situation command ring transition interrupts will not occur since the class driver will never fill the command ring.

The Ownership Bit

The ownership bit 278 in each ring entry is like the flag on an old-fashioned mailbox. The postman raised the flag to indicate that a letter had been put in the box. When the box was emptied, the owner would lower the flag. Similarly, the ownership bit indicates that a message has been deposited in a ring entry, and whether or not the ring entry (i.e., mailbox) has been emptied. Once a message is written to a ring entry, that message must be emptied before a second message can be written over the first.

For a command descriptor, the ownership bit "0" is changed from zero to one when the host has filled the descriptor and is releasing it to the port. Conversely, once the port has emptied the command descriptor and is returning the empty slot to the host, the ownership bit is changed from one to zero. That is, to send a command the host sets the ownership bit to one; the port clears it when the command has been received, and returns the empty slot to the host.

To guarantee that the port/controller sees each command in a timely fashion, whenever the host inserts a command in the command ring, it must read the IP register. This forces the port to poll if it was not already polling.

For a response descriptor, when the ownership bit 0 undergoes a transition from one to zero, that means that the port has filled the descriptor and is releasing it to the host. The reverse transition

means that the host has emptied the response descriptor and is returning the empty slot to the port. Thus, to send a response the port clears the ownership bit, while the host sets it when the response has been received, and returns the empty slot to the port.

Just as the port must poll for commands, the host must poll for responses, particularly because of the possibility of unsolicited responses.

Interrupts

The transmission of a message will result in a host interrupt if and only if interrupts were armed (i.e., enabled) suitably during initialization and one of the following three conditions has been met: (1) the message was a command with flag 280 equal to one (i.e., F = 1), and the fetching of the command by the port caused the command ring to undergo a transition from full to not-full; (2) if the message was a response with F = 1 and the depositing of the message by the port caused the response ring to make a transition from empty to not-empty; or (3) the port is interfaced to the host via a bus adapter and a command required the port/controller to re-access a given location during data transfer. (The latter interrupt means that the port/controller is requesting the host to purge the indicated channel of the bus adapter.)

Port Polling

The reading of the IP register by the host causes the port/controller to poll for commands. The port/controller begins reading commands out of host memory; if the controller has an internal command buffering capability, it will write commands into the buffer if they can't be executed immediately. The port continues to poll for full command slots until the command ring is found to be empty, at which time it will cease polling. The port will resume polling either when the controller delivers a response to the host, or when the host reads the IP register.

Correspondingly, response polling for empty slots continues until all commands buffered within the controller have been completed and the associated responses have been sent to the host.

Host Polling

Since unsolicited responses are possible, the host cannot cease polling for responses when all outstanding commands have been acknowledged, though. If it did, an accumulation of unsolicited messages would first saturate the response ring and then any controller internal message buffers, blocking the controller and preventing it from processing additional commands. Thus, the host must at least occassionally scan the response ring, even when not expecting a response. One way to accomplish this is by using the ring transition interrupt facility described above; the host also would remove in sequence from the response ring as many responses as it finds there.

Data Transmission

Data transmission details are controller-dependent. There are certain generic characteristics, however.

Data transfer commands are assumed to contain buffer descriptors and byte or word counts. The buffers serve as sources or sinks for the actual data transfers, which are effected by the port as non-processor (NPR or DMA) transfers under command-derived count control to or from the specified buffers. A buffer descriptor begins at the first word allocated for this purpose in the formats of higher-level commands. When used with the UNIBUS interconnection, the port employs a two-word buffer descriptor format as illustrated in Fig. 10. As shown herein, the bits in the low-order buffer address 292 are message-dependent. The bits labelled "U" (294, 296) in the high-order portion 298 of the buffer descriptor are the high-order bits of an 18-bit UNIBUS address. The bits 300-306, labelled "Q", are usable as an extension to the high-order UNIBUS address, and are zero for UNIBUS systems.

Repeated access to host memory locations must be allowed for both read and write operation, in random sequence, if the interfaces are to support higher-level protocol functions such as transfer restarts, compares, and so forth. In systems with buffered bus adapters, which require a rigid sequencing, this necessitates purging of the relevant adapter channel prior to changing from read to write, or vice versa, and prior to breaking an addressing sequence. Active cooperation of the host CPU is required for this action. The port signals its desire for an adapter channel purge, as indicated above under the heading "The Communications Area". The host performs the purge and writes zeros to the SA register 38 to signal completion.

Transmission Errors

Four classes of transmission errors have been considered in the design of this interface: (1) failure to become bus mater; (2) failure to become interrupt master; (3) bus data timeout error; and (4) bus parity error.

When the port (controller) attempts to access host memory, it must first become the "master" of bus 60. To deal cleanly with the possibility of this

exercise failing, the port sets up a corresponding "last fail" response packet (see below) before actually requesting bus access. Bus access is then requested and if the port timer expires, the host will reinitialize the port/controller. The port will then report the error via the "last fail" response packet (assuming such packets were enabled during the reinitialization).

A failure to become interrupt master occurs whenever the port attempts to interrupt the host and an acknowledgment is not forthcoming. It is treated and reported the same as a failure to become bus master, although the contents of its last fail response will, of course, be different.

Bus data timeout errors involve failure to complete the transfer of control or data messages. If the controller retires a transfer after it has failed once, and a second try also fails, then action is taken responsive to the detection of a persistent error. If the unsuccessful operation was a control transfer, the port writes a failure code into the SA register and then terminates the connection with the host. Naturally, the controller will have to be reinitialized. On the other hand, if the unsucessful operation was a data transfer, the port/controller stays online to the host and the failure is reported to the host in the response packet for the involved operation. Bus parity errors are handled the same as bus data timeout errors.

Fatal Errors

Various fatal errors may be self-detected by the port or controller. Some of these may also arise while the controller is operating its attached peripheral device(s). In the event of a fatal error, the port sets in the SA register a one in its most significant bit, to indicate the existence of a fatal error, and a fatal error code in bits 10-0.

Interrupt Generation Rate

Under steady state conditions, at most one ring interrupt will be generated for each operation (i.e., command or response transmission). Under conditions of low I/O rate, this will be due to response ring transitions from empty to not-empty; with high I/O rate, it will be due to command ring transitions from full to not-full. If the operation rate fluctuates considerably, the ratio of interrupts to operations can be caused to decline from one-to-one. For example, an initially low but rising operation rate will eventually cause both the command and response rings to be partially occupied, at which point interrupts will cease and will not resume until the command ring fills and begins to make full to not-full transitions. This point can be staved off by increasing the permissible depth of the command ring. Generally, the permissible depth of the response ring will have to be increased also, since saturation of the response ring will eventully cause the controller to be unwilling to fetch additional commands. At that point, the command queue will saturate and each fetch will generate an interrupt.

Moreover, a full condition in either ring implies that the source of that ring's entries is temporarily choked off. Consequently, ring sizes should be large enough to keep the incidence of full rings small. For the command ring, the optimal size depends on the latency in the polling of the ring by the controller. For the response ring, the optimal size is a function of the latency in the ring-emptying software.

Initialization

A special initialization procedure serves to (1) identify the parameters of the host-resident communications region to the port; (2) provide a confidence check on port/controller integrity; and (3) bring the port/controller online to the host.

The initialization process starts with a "hard" initialization during which the port/controller runs some preliminary diagnostics. Upon successful completion of those diagnostics, there is a four step procedure which takes place. First, the host tells the controller the lengths of the rings, whether initialization interrupts are to be armed (i.e., enabled) and the address(es) of the interrupt vector(s). The port/controller then runs a complete internal integrity check and signals either success or failure. Second, the controller echos the ring lengths, and the host sends the low-order portion of the ring-base address and indicates whether the host is one which requires purge interrupts. Third, the controller sends an echo of the interrupt vector address(es) and the initialization interrupt arming signal. The host then replies with the high-order portion of the ringbase address, along with a signal which conditionally triggers an immediate test of the polling and adapter purge functions of the port. Fourth, the port tests the ability of the input/output bus to perform nonprocessor (NPR) transfers. If successful, the port zeros the entire communications area and signals the host that initialization is complete. The port then awaits a signal from the host that the controller should begin normal operation.

At each step, the port informs the host of either success or failure. Success leads to the next initialization step and failure causes a restart of the initialization sequence. The echoing of information to the host is used to check all bit positions in the transport mechanism and the IP and SA registers.

The SA register is heavily used during initialization. The detailed format and meaning of its

contents depend on the initialization step involved and whether information is being read from or written into the register. When being read, certain aspects of the SA format are constant and apply to all steps. This constant SA read format is indicated in Fig. 11. As seen there, the meaning of bits 15-11 of SA register 38 is constant but the interpretation of bits 10-0 varies. The S4-S1 bits, 316-310, are set separately by the port to indicate the initialization step number which the port is ready to perform or is performing. The S1 bit 310 is set for initialization step 1; the S2 bit 312, for initialization step 2, etc. If the host detects more than one of the S1-S4 bits 316-310 set at any time, it restarts the initialization of the port/controller; the second time this happens, the port/controller is presumed to be malfunctioning. The SA register's most significant bit 318, labelled ER, normally is zero; it may be set to the value of 1 if either a port/controller-based diagnositic test has failed, or there has been a fatal error. In the event of such a failure or error, bits 10-0 comprise a field 320 into which an error code is written; the error code may be either port-generic or controller-dependent. Consequently, the host can determine not only the nature of an error but also the step of the initialization during which it occurred. If a fatal error is detected during hard initialization, prior to the start of initialization step 1, the ER bit is set to a value of 1 and no step bit is set.

The occurrence of an initialization error causes the port driver to retry the initialization sequence at least once.

Reference will now be made to Figs. 12A-12D, wherein the details of the initialization process are illustrated.

The host begins the initialization sequence either by performing a hard initialization of the controller (this is done either by issuing a bus initialization (INIT) command (Step 322)) or by writing zeros to the IP register. The port guarantees that the host reads zeros in the SA register on the next bus cycle. The controller, upon sensing the initialization order, runs a predetermined set of diagnostic routines intended to ensure the minimum integrity necessary to rely on the rest of the sequence. (Step 324) Initialization then sequences through the four above-listed steps.

At the beginning of each initialization step n, the port clears bit Sn-1 before setting bit Sn; thus, the host will never see bits Sn-1 and Sn set simultaneously. From the viewpoint of the host, step n begins when reading the SA register results in the transition of bits S_n from 0 to 1. Each step ends when the next step begins, and an interrupt may accompany the step change if interrupts are enabled.

Each of initialization steps 1 - 3 is timed and if any of those steps fails to complete within the alloted time, that situation is treated as a host-detected fatal error. By contrast, there is no explicit signal for the completion of initialization step 4; rather, the host observes either that controller operation has begun or that a higher-level protocol-dependent timer has expired.

The controller starts initialization step 1 by writing to the SA register 38 the pattern indicated in Fig. 12A. (Step 326) Bits 328-332 are controller-dependent. The "NV" bit, 332, indicates whether the port supports a host-settable interrupt vector address; a bit value of 1 provides a negative answer. The "QB" bit, 330, indicates whether the port supports a 22-bit host bus address; a 1 indicates an affirmative answer. The "DI", bit 328, indicates whether the port implements enhanced diagnostics, such as wrap-around, purge and poll test; an affirmative answer is indicated by a bit value of 1.

The host senses the setting of bit 310, the S1 bit, and reads the SA register. (Step 334.) It then responds by writing into the SA register the pattern shown in step 336. The most significant bit 338 in the SA register 38 is set to a 1, to guarantee that the port does not interpret the pattern as a host "adapter purge complete" response (after a spontaneous reinitialization). The WR bit, 340, indicates whether the port should enter a diagnostic wrap mode wherein it will echo messages sent to it; a bit value of 1 will cause the port to enter that mode. The port will ignore the WR bit if DI = 0 at the beginning of initialization step 1. Field 342, comprising bits 13-11 and labelled "C RNG LNG," indicates the number of entries or slots in the command ring, expressed as a power of 2. Similarly, field 344, comprising bits 10-8 and labelled "R RNG LNG", represents the number of response ring slots (i.e., the length of the response ring), also expressed as a power of 2. Bit 346, the number 7 bit in the register, labelled "IE", indicates whether the host is arming interrupts at the completion of each of steps 1 - 3. An affirmative answer is indicated by a 1. Finally, field 348, comprising register bits 6-0, labelled "INT Vector", contains the address of the vector to which all interrupts will be directed, divided by 4. If this address is 0, then port interrupts will not be generated under any circumstances. If this field is non-zero the controller will generate initialization interrupts (if IE is set) and purge interrupts (if PI is set), and ring transition interrupts depending on the FLAG bit setting of the ring entry causing the transition.

The port/controller reads the SA register after it has been written by the host and then begins to run its full integrity check diagnostics; when finished, it conditionally interrupts the host as described above. (Step 350).

This completes step 1 of the initialization process. Next, the controller writes a pattern to the SA register as indicated in Fig. 12B. (Step 352.) As shown there, bits 7-0 of the SA register echo bits 15-8 in step 336. The response and command ring lengths are echoed in fields 354 and 356, respectively; bit 358 echoes the host's WR bit and bit 360 echoes the host's bit 15. The port type is indicated in field 362, register bits 10-8, and bit 12 is set to a 1 to indicate the beginning of step 2.

The host reads the SA register and validates the echo when it sees bit S2 change state. (Step 364.) If everything matches up, the host then responds by writing into THE SA register the pattern indicated in step 366. Field 368, comprising SA register bits 15-1, labelled "ringbase lo address", represents the low-order portion of the address of the word [ringbase + 0] in the communications area. While this is a 16-bit byte address, its lowest order bit is 0, implicitly. The lowest order bit of the SA register, 370, indicated as "PI", when set equal to 1, means that the host is requesting adapter purge interrupts.

The controller reads the low ringbase address (step 372) and then writes into the SA register the pattern indicated in step 374, which starts initialization step 3 by causing bit 376, the S3 bit, to undergo a transition from 0 to 1. The interrupt vector field 348 and interrupt enabling bit 346 from step 336 are echoed in SA register bits 7-0.

Next, the host reads the SA register and validates the echo; if the echo did not operate properly, an error is signalled. (Step 378). Assuming the echo was valid, the host then writes to the SA register the pattern indicated in step 380. Bit 382, the most significant bit, labelled "PP", is written with an indication of whether the host is requesting execution of "purge" and "poll" tests (described elsewhere); an affirmative answer is signaled by a 1. The port will ignore the PP bit if the DI bit 328 was zero at the beginning of step 1. The "ringbase hi address" field 384, comprising SA register bits 14-0, is the high-portion of the address [ringbase + 0].

The port then reads the SA register; if the PP bit has been set, the port writes zeroes into the SA register, to signal its readiness for the test. (Step 386). The host detects that action and itself writes zeroes (or anything else) to the SA register, to simulate a "purge completed" host action. (Step 388.) After the port verifies that the host has written to the SA register (Step 390.), the host reads, and then disregards, the IP register. Step 392.) This simulates a "start polling" command from the host to the port. The port verifies that the IP register was read, step 394, before the sequence continues. The host is given a predetermined time from the time the SA register was first written during initialization step 3 within which to complete these actions. (Step 396) If it fails to do so, initialization stops. The host may then restart the initialization sequence from the beginning.

Upon successful completion of initialization step 3, the transition to initialization step 4 is effectuated when the controller writes to the SA register the pattern indicated in step 398. Field 400, comprising bits 7-0 of the SA register, contains the version number of the port/controller microcode. In a microprogrammed controller, the functionality of the controller can be altered by changing the programming. It is therefore important that the functionality of the host with the ability to recognize which versions of the controller microcode are compatible with the host and which are not. Therefore, the host checks the controller microcoder version in field 400 and confirms that the level of functionality is appropriate to that particular host. (Step 402.) The host responds by writing into the SA register the pattern indicated in step 404. It is read by the controller in step 405 and 406 and the operational microcode is then started.

The "burst" field in bits 7-2 of the SA register is one less than the maximum number of longwords the host is willing to allow per NPR (nonprocessor involved) transfer. The port uses a default burst count if this field is zero. The values of both the default and the maximum the port will accept are controller-dependent. If the "LF" bit 408 is set equal to 1, that indicates that the host wants a "last fail" response packet when initialization is completed. The state of the LF bit 408 does not have any effect on the enabling/disabling of unsolicited responses. The meaning of "last fail" is explained below. The "GO" bit 410 indicates whether the controller should enter its functional microcode as soon as initialization completes. If GO = 0, when initialization completes, the PORT will continue to read the SA register until THE host forces bit 0 of that register to make the transition from 0 to 1.

At the end of initialization step 4, there is no explicit interrupt request. Instead, if interrupts were enabled, the next interrupt will be due either to a ring transition or to an adapter purge request.

Diagnostic Wrap Mode

Diagnostic Wrap Mode (DWM) provides host-based diagnostics with the means for verifying the lowest levels of host-controller communication via the port. In DWM, the port attempts to echo in the SA register 38 any data written to that register by the host. DWM is a special path through initialization step 1; initialization steps 2-4 are suppressed and the port/controller is left disconnected from the host. A hard initialization terminates DWM and, if

the results of DWM are satisfactory, it is then bypassed on the next initialization sequence.

Last Fail

"Last fail" is the name given to a unique response packet which is sent to THE HOST WHEN the port/controller detected an error during a previous "run" and the LF bit 405 was set in step 404 of the current initialization sequence. It is sent when initialization completes. The format of this packet is indicated in Fig. 13. The packet starts with 64 bits of zeroes in a pair of 32 bit words 420. Next there is a 32 bit word 422 consisting of a lower-order byte 422A and a higher-order byte 422B, each of which has unique numerical contents. Word 422 is followed by a double word 424 which contains a controller identifier. The packet is concluded by a single word 426. The higher-order byte 426A of word 426 contains an error code. The lower half of word 426 is broken into a pair of 8 bit fields 426B and 426C. Field 426B contains the controller's hardware revision number. Field 426C contains the controller's software, firmware or microcode revision number.

Recap

It should be apparent from the foregoing description that the present invention provides a versatile and powerful interface between host computers and peripheral devices, particularly secondary mass storage subsystems. This interface supports asynchronous packet type command and response exchanges, while obviating the need for a hardware-interlocked bus and greatly reducing the interrupt load on the host processor. The efficiency of both input/output and processor operation are thereby enhanced.

A pair of registers in the controller are used to transfer certain status, command and parametric information between the peripheral controller and host. These registers are exercised heavily during a four step initialization process. The meanings of the bits of these registers change according to the step involved. By the completion of the initialization sequence, every bit of the two registers has been checked and its proper operation confirmed. Also, necessary parametric information has been exchanged (such as ring lengths) to allow the host and controller to communicate commands and responses.

Although the host-peripheral communications interface of the invention comprises a port which, effectively, is controller-based, it nevertheless is largely localized at the host. Host-side port elements include: the command and response rings; the ring transition indicators; and, if employed, bus data purge control. At the controller, the port elements include: command and response buffers, host command and response ring pointers, and the SA and IP registers.

Claims

1. A method of initializing a data processing system which includes a host processor (70), mass storage control means (30), a bus means (60) for coupling the host processor (70) and the mass storage control means (30), a status and address register (38) coupled to the bus means (60), a mass storage means (40) and a cable (50) for coupling the mass storage means (40) to the mass storage control means (30), the register (38) having multiple bit locations for storing values, said method being characterized by the steps of:
writing, by the mass storage control means (30) to the multiple bit locations of the status and address register (38) a predetermined first bit pattern thereby setting a predetermined first bit location (SA (11)) of the status and address register (38) to indicate the beginning of a first initialization step;
sensing, by the host processor (70), the predetermined first bit location of the status and address register (38), and if set, reading the predetermined first bit pattern in the register (38);
writing, by the host processor (70), a predetermined second bit pattern to the multiple bit locations of the status and address register (38) indicative of at least the length of each of a command ring and a response ring;
reading, by the mass storage control means (30), the predetermined second bit pattern in the register (38) and initiating diagnostic activities in the mass storage control means (30);
writing, by the mass storage control means (30), a predetermined third bit pattern to the multiple bit locations of the status and address register (38), indicative of at least the length of each of the command ring and the response ring, thereby setting a predetermined second bit location (SA (12)) in the status and address register (38) to indicate the beginning of a second initialization step;
sensing, by the host processor (70), the predetermined second bit location of the status and address register (38), and if set, reading the predetermined third bit pattern in the register (38);
writing, by the host processor (70), a predetermined fourth bit pattern to the multiple bit locations of the status and address register (38) corresponding to at least a portion of a

ringbase address;
reading, by the mass storage control means (30), the predetermined fourth bit pattern in the register (38);
writing, by the mass storage control means (30), a predetermined fifth bit pattern to the multiple bit location of the status and address register (38) thereby setting a predetermined third bit location (SA (13)) of the status and address register (38) to indicate the beginning of a third initialization step;
sensing, by the host processor (70), the predetermined third bit location of the status and address register (38) and if set, reading the predetermined fifth bit pattern in the register (38);
writing, by the host processor (70), a predetermined sixth bit pattern to the multiple bit locations of the status and address register (38) corresponding to at least another portion of the ringbase address;
reading, by the mass storage control means (30), the predetermined sixth bit pattern in the register (38);
writing, by the mass storage control means (30), to the multiple bit locations of the status and address register (38), a predetermined seventh bit pattern corresponding to, at least, a microcode version of the mass storage control means (30) thereby setting a predetermined fourth bit location (SA (14)) of the status and address register (38) to indicate the beginning of a fourth initialization step;
sensing, by the host processor (70), the predetermined fourth bit location of the status and address register (38), and if set, reading the predetermined seventh bit pattern in the register (38); and
determining in the host processor (70) whether the microcode version is compatible with the host processor (70).

2. The method of claim 1 further including the steps of:
timing the reading, writing and sensing operations; and
determining an error condition if the aggregate time to perform the reading, writing and sensing operations exceeds a preselected time.

3. The method of claim 1 further including the steps of:
comparing a preselected portion of the predetermined second bit pattern with a preselected portion of the predetermined third bit pattern;
and
determining an error condition if the comparing of preselected portions of the predetermined second and third bit patterns does not yield a match.

Patentansprüche

1. Verfahren zum Initialisieren eines Datenverarbeitungssystems, das folgendes enthält: einen Hostprozessor (70), eine Massenspeicher-Steuereinrichtung (30), eine Buseinrichtung (60) zum Koppeln des Hostprozessors (70) und der Massenspeicher-Steuereinrichtung (30), ein Zustands- und Adreßregister (38), das mit der Buseinrichtung (60) gekoppelt ist, eine Massenspeichereinrichtung (40) und ein Kabel (50) zum Koppeln der Massenspeichereinrichtung (40) mit der Massenspeicher-Steuereinrichtung (30), wobei das Register (38) Mehrbit-Stellen zum Speichern von Werten aufweist, wobei das Verfahren durch die folgenden Schritte gekennzeichnet ist:
Schreiben eines vorbestimmten ersten Bitmusters durch die Massenspeicher-Steuereinrichtung (30) an die Mehrbit-Stellen des Zustands- und Adreßregisters (38), wodurch eine vorbestimmte erste Bitstelle (SA (11)) des Zustands- und Adreßregisters (38) gesetzt wird, um den Anfang eines ersten Initialisierungsschritts anzuzeigen;
Abtasten der vorbestimmten ersten Bitstelle des Zustands- und Adreßregisters (38) durch den Hostprozessor (70), und wenn sie gesetzt ist, Lesen des vorbestimmten ersten Bitmusters in dem Register (38);
Schreiben eines vorbestimmten zweiten Bitmusters an die Mehrbit-Stellen des Zustands- und Adreßregisters (38) durch den Hostprozessor (70), das wenigstens die Länge sowohl eines Befehlsrings als auch eines Antwortrings anzeigt;
Lesen des vorbestimmten zweiten Bitmusters in dem Register (38) durch die Massenspeicher-Steuereinrichtung (30) und Initiieren von Diagnoseaktivitäten in der Massenspeicher-Steuereinrichtung (30);
Schreiben eines vorbestimmten dritten Bitmusters an die Mehrbit-Stellen des Zustands- und Adreßregisters (38) durch die Massenspeicher-Steuereinrichtung (30), das wenigstens die Länge sowohl des Befehlsrings als auch des Antwortrings anzeigt, wodurch eine vorbestimmte zweite Bitstelle (SA (12)) in dem Zustands- und und Adreßregister (38) gesetzt wird, um den Beginn eines zweiten initialisierungsschritts anzuzeigen;
Abtasten der vorbestimmten zweiten Bitstelle des Zustands- und Adreßregisters (38) durch den Hostprozessor (70), und wenn sie gesetzt

ist, Lesen des vorbestimmten dritten Bitmusters in dem Register (38);
Schreiben eines vorbestimmten vierten Bitmusters an die Mehrbit-Stellen des Zustands- und Adreßregisters (38) durch den Hostprozessor (70) entsprechend wenigstens einem Teil einer Ringbasisadresse;
Lesen des vorbestimmten vierten Bitmusters in dem Register (38) durch die Massenspeicher-Steuereinrichtung (30);
Schreiben eines vorbestimmten fünften Bitmusters an die Mehrbit-Stellen des Zustands- und Adreßregisters (38) durch die Massenspeicher-Steuereinrichtung (30), wodurch eine vorbestimmte dritte Bitstelle (SA (13)) des Zustands- und Adreßregisters (38) gesetzt wird, um den Beginn eines dritten Initialisierungsschritts anzuzeigen;
Abtasten der vorbestimmten dritten Bitstelle des Zustands- und Adreßregisters (38) durch den Hostprozessor (70), und wenn sie gesetzt ist, Lesen des vorbestimmten fünften Bitmusters in dem Register (38);
Schreiben eines vorbestimmten sechsten Bitmusters an die Mehrbit-Stellen des Zustands- und Adreßregisters (38) durch den Hostprozessor (70) entsprechend wenigstens einem weiteren Teil der Ringbasisadresse;
Lesen des vorbestimmten sechsten Bitmusters in dem Register (38) durch die Massenspeicher-Steuereinrichtung (30);
Schreiben eines vorbestimmten siebten Bitmusters an die Mehrbit-Stellen des Zustands- und Adreßregisters (38) durch die Massenspeicher-Steuereinrichtung (30) entsprechend wenigstens einer Mikrocode-Version der Massenspeicher-Steuereinrichtung (30), wodurch eine vorbestimmte vierte Bitstelle (SA (14)) des Zustands- und und Adreßregisters (38) gesetzt wird, um den Beginn eines vierten Initialisierungsschritts anzuzeigen;
Abtasten der vorbestimmten vierten Bitstelle des Zustands- und Adreßregisters (38) durch den Hostprozessor (70), und wenn sie gesetzt ist, Lesen des vorbestimmten siebten Bitmusters in dem Register (38); und
Bestimmen in dem Hostprozessor (70), ob die Mikrocode-Version kompatibel zu dem Hostprozessor (70) ist.

2. Verfahren nach Anspruch 1, das weiterhin die folgenden Schritte enthält:
zeitliche Abstimmung der Lese-, Schreib- und Erfassungsoperationen; und
Bestimmen eines Fehlerzustands, wenn die angehäufte Zeit zum Durchführen der Lese-, Schreib- und Erfassungsoperationen eine vorher ausgewählte Zeit überschreitet.

3. Verfahren nach Anspruch 1, das weiterhin die folgenden Schritte enthält:
Vergleichen eines vorbestimmten Teils des vorbestimmten zweiten Bitmusters mit einem zuvor ausgewählten Teil des vorbestimmten dritten Bitmusters; und
Bestimmen eines Fehlerzustands, wenn das Vergleichen der zuvor ausgewählten Teile des vorbestimmten zweiten und des vorbestimmten dritten Bitmusters keine Anpassung ergibt.

Revendications

1. Procédé d'initialisation d'un système de traitement de données qui comprend un ordinateur central (70), des moyens de commande de mémoire de masse (30), des moyens formant bus (60) pour coupler l'ordinateur central (70) et les moyens de commande de mémoire de masse (30), un registre d'état et d'adresses (38) couplé aux moyens formant bus (60), des moyens formant mémoire de masse (40) et un câble (50) pour coupler les moyens formant mémoire de masse (40) aux moyens de commande de mémoire de masse (30), le registre (38) ayant des emplacements de bits multiples pour emmagasiner des valeurs, ledit procédé étant caractérisé par les étapes consistant:

à écrire, à l'aide des moyens de commande de mémoire de masse (30), dans les emplacements de bits multiples du registre d'état et d'adresses (38) une première configuration binaire prédéterminée, mettant ainsi à un un premier emplacement de bits prédéterminé (SA (11)) du registre d'état et d'adresses (38) pour indiquer le début d'une première étape d'initialisation;

à détecter, à l'aide de l'ordinateur central (70), le premier emplacement de bits prédéterminé du registre d'état et d'adresses (38) et, s'il est à un, à lire la première configuration binaire prédéterminée dans le registre (38);

à écrire à l'aide de l'ordinateur central (70), une seconde configuration binaire prédéterminée dans les emplacements de bits multiples du registre d'état et d'adresses (38) indiquant au moins la longueur aussi bien d'un anneau de commande que d'un anneau de réponse;

à lire, à l'aide des moyens de commande de mémoire de masse (30), la seconde configuration binaire prédéterminée dans le registre (38) et à déclencher les activités de diagnostic dans les moyens de commande de mémoire de masse (30);

à écrire, à l'aide des moyens de commande de mémoire de masse (30), une troisième configuration binaire prédéterminée dans les

emplacements de bits multiples du registre d'état et d'adresses (38), indiquant au moins la longueur aussi bien de l'anneau de commande que de l'anneau de réponse, mettant ainsi à un un second emplacement de bits prédéterminé (SA (12)) du registre d'état et d'adresses (38) pour indiquer le début d'une seconde étape d'initialisation;

à détecter, à l'aide de l'ordinateur central (70), le second emplacement de bits prédéterminé du registre d'état et d'adresses (38) et, s'il est à un, à lire la troisième configuration binaire prédéterminée dans le registre (38);

à écrire, à l'aide de l'ordinateur central (70), une quatrième configuration binaire prédéterminée dans les emplacements de bits multiples du registre d'état et d'adresses (38) correspondant à au moins une partie d'une adresse de base annulaire;

à lire, à l'aide des moyens de commande de mémoire de masse (30), la quatrième configuration binaire prédéterminée dans le registre (38);

à écrire, à l'aide des moyens de commande de mémoire de masse (30), une cinquième configuration binaire prédéterminée dans les emplacements de bits multiples du registre d'état et d'adresses (38), mettant ainsi à un un troisième emplacement de bits prédéterminé (SA (13)) du registre d'état et d'adresses (38) pour indiquer le début d'une troisième étape d'initialisation;

à détecter, à l'aide de l'ordinateur central (70), le troisième emplacement de bits prédéterminé du registre d'état et d'adresses (38) et, s'il est à un, à lire la cinquième configuration de bits prédéterminé dans le registre (38);

à écrire, à l'aide l'ordinateur central (70), une sixième configuration binaire prédéterminée dans les emplacements de bits multiples du registre d'état et d'adresses (38) correspondant à au moins une autre partie de l'adresse de base annulaire;

à lire, à l'aide des moyens de commande de mémoire de masse, la sixième configuration binaire prédéterminée dans le registre (38);

à écrire, à l'aide des moyens de commande de mémoire de masse (30), dans les emplacements de bits multiples du registre d'état et d'adresses (38), une septième configuration binaire prédéterminée correspondant à, au moins, une version microcode des moyens de commande de mémoire de masse (30), mettant ainsi à un un quatrième emplacement de bits prédéterminé (SA (14)) du registre d'état et d'adresses (38) pour indiquer le début d'une quatrième étape d'initialisation;

à détecter, à l'aide de l'ordinateur central (70), le quatrième emplacement de bits prédéterminé du registre d'état et d'adresses (38) et, s'il est à un, à lire la septième configuration binaire prédéterminée dans le registre (38); et

à déterminer, dans l'ordinateur central (70), si la version microcode est compatible avec l'ordinateur central (70).

2. Procédé selon la revendication 1 comprenant, en outre, les étapes consistant:

à minuter les opérations de lecture, d'écriture et de détection; et

à déterminer la présence d'une condition d'erreur si le temps total pris par l'exécution des opérations de lecture, d'écriture et de détection dépasse une durée présélectionnée.

3. Procédé selon la revendication 1 comprenant, en outre, les étapes consistant:

à comparer une partie préselectionnée de la seconde configuration binaire prédéterminée à une partie présélectionnée de la troisième configuration binaire prédéterminée; et

à déterminer la présence d'une condition d'erreur si la comparaison des parties présélectionnées des seconde et troisième configurations binaires prédéterminées ne fait pas ressortir une concordance.

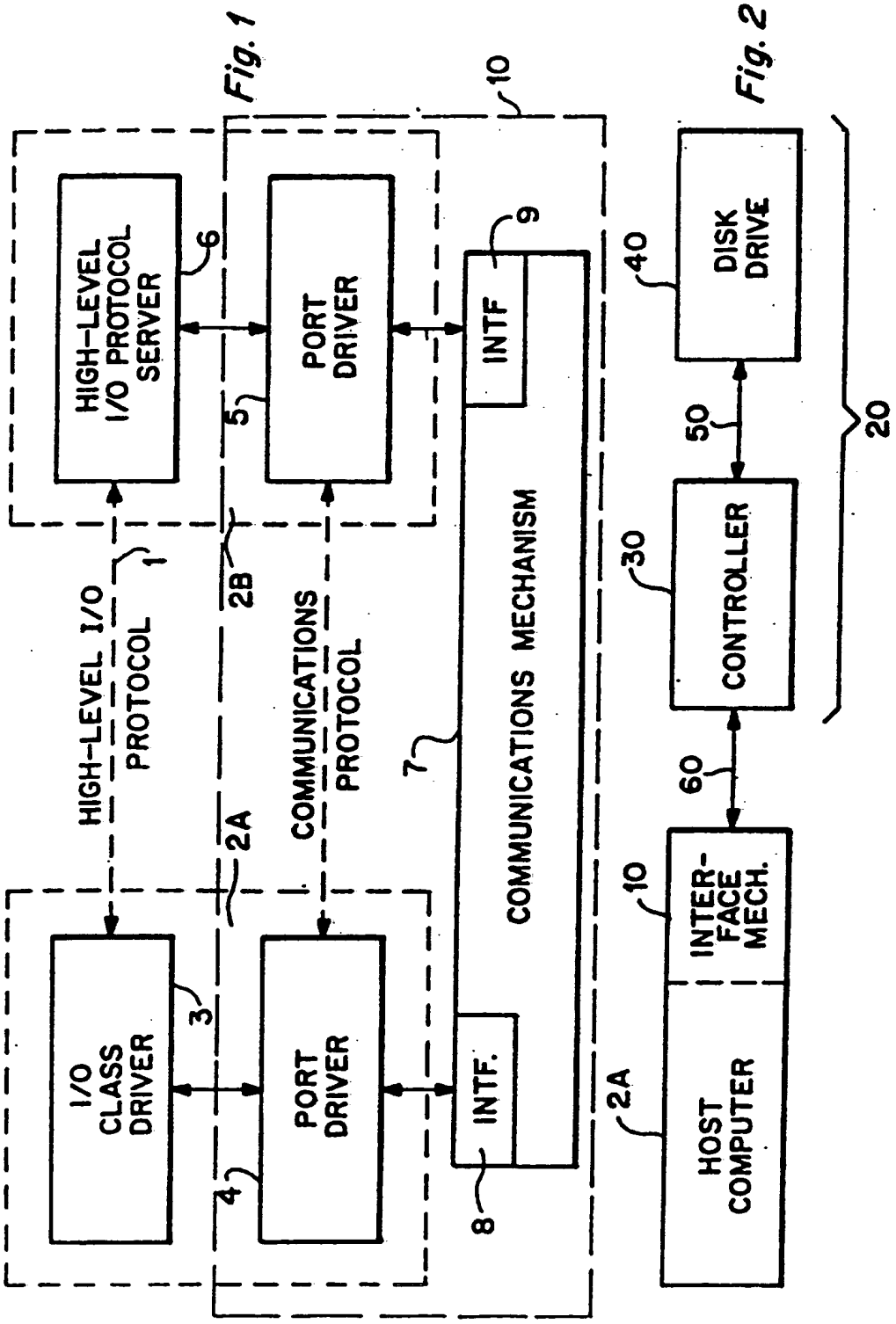
HIGH-LEVEL I/O PROTOCOL
1
HIGH-LEVEL I/O PROTOCOL SERVER
6
PORT DRIVER
5
2B
COMMUNICATIONS PROTOCOL
2A
I/O CLASS DRIVER
3
PORT DRIVER
4
INTF.
8
7
COMMUNICATIONS MECHANISM
INTF
9
10
10
INTER-FACE MECH.
2A
HOST COMPUTER
60
30
CONTROLLER
50
40
DISK DRIVE
20

Fig. 1

Fig. 2

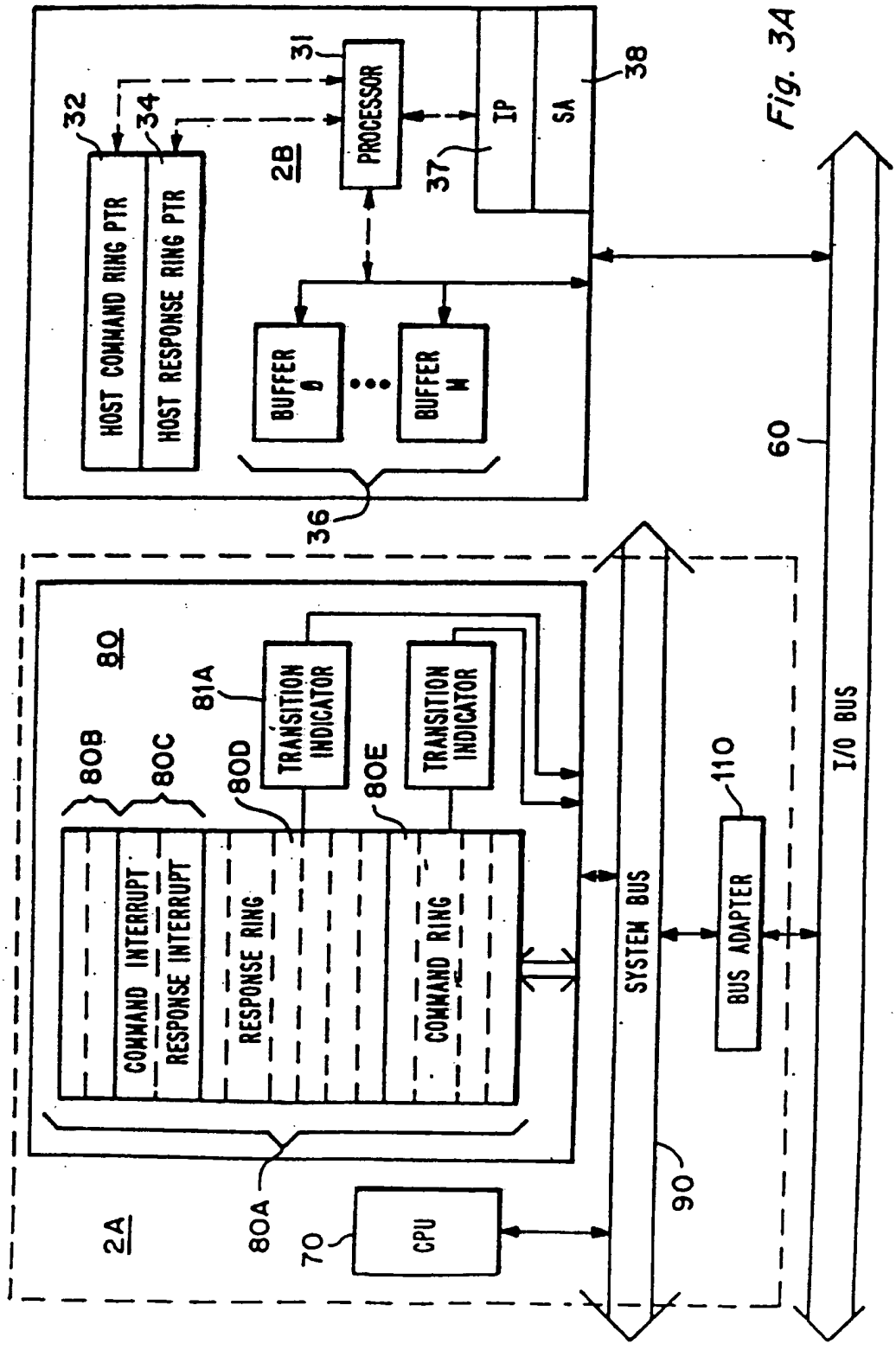
2A
2B
80
80A
80B
80C
80D
80E
81A
32
34
31
36
37
38
60
70
90
110
HOST COMMAND RING PTR
HOST RESPONSE RING PTR
PROCESSOR
IP
SA
BUFFER 0
BUFFER M
TRANSITION INDICATOR
TRANSITION INDICATOR
COMMAND INTERRUPT
RESPONSE INTERRUPT
RESPONSE RING
COMMAND RING
CPU
SYSTEM BUS
BUS ADAPTER
I/O BUS

Fig. 3A

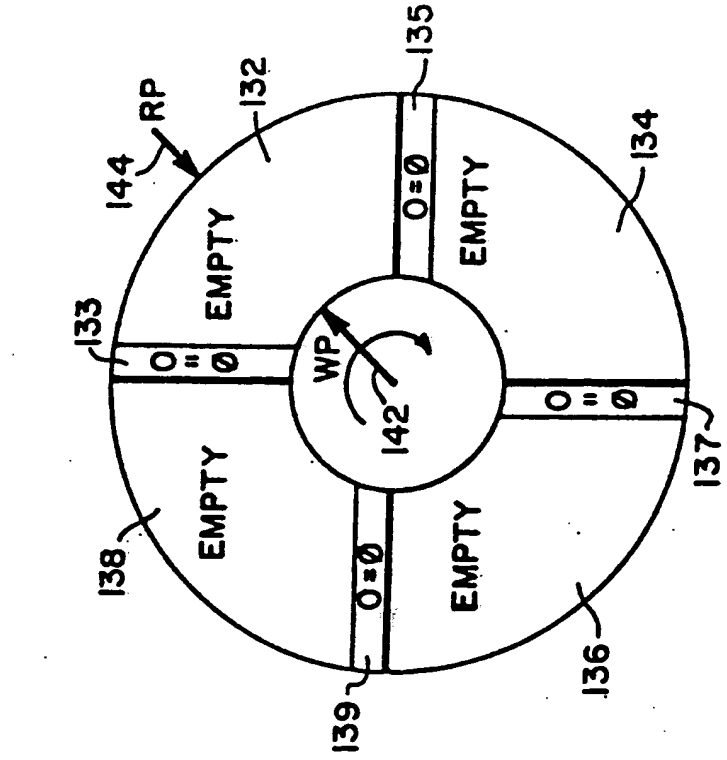
RP
144
132
135
134
133
EMPTY
O=Ø
EMPTY
O=Ø
WP
142
O=Ø
137
EMPTY
EMPTY
138
O=Ø
139
136

Fig. 3C

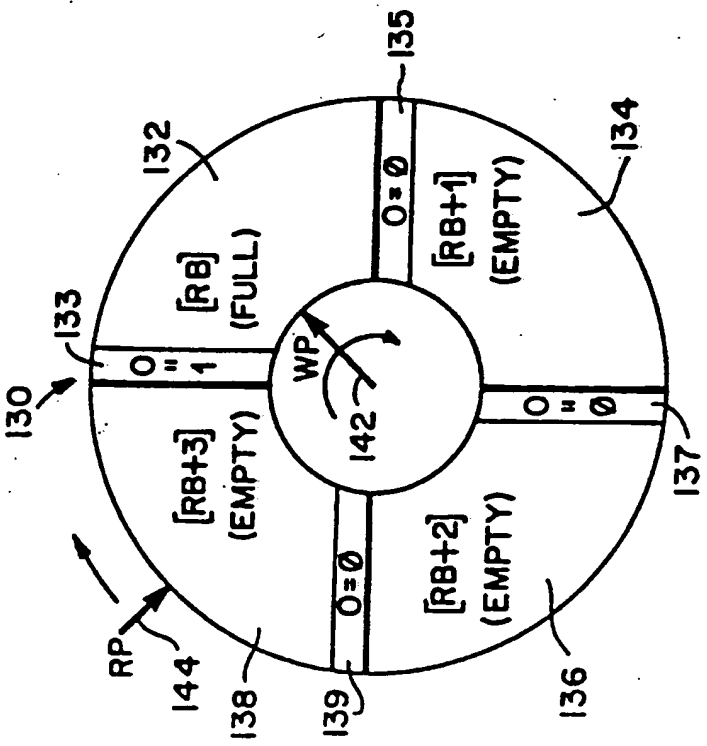
130
RP
144
132
135
134
133
[RB]
(FULL)
O=Ø
[RB+1]
(EMPTY)
O=1
WP
142
O=Ø
137
[RB+3]
(EMPTY)
[RB+2]
(EMPTY)
O=Ø
138
139
136

Fig. 3B

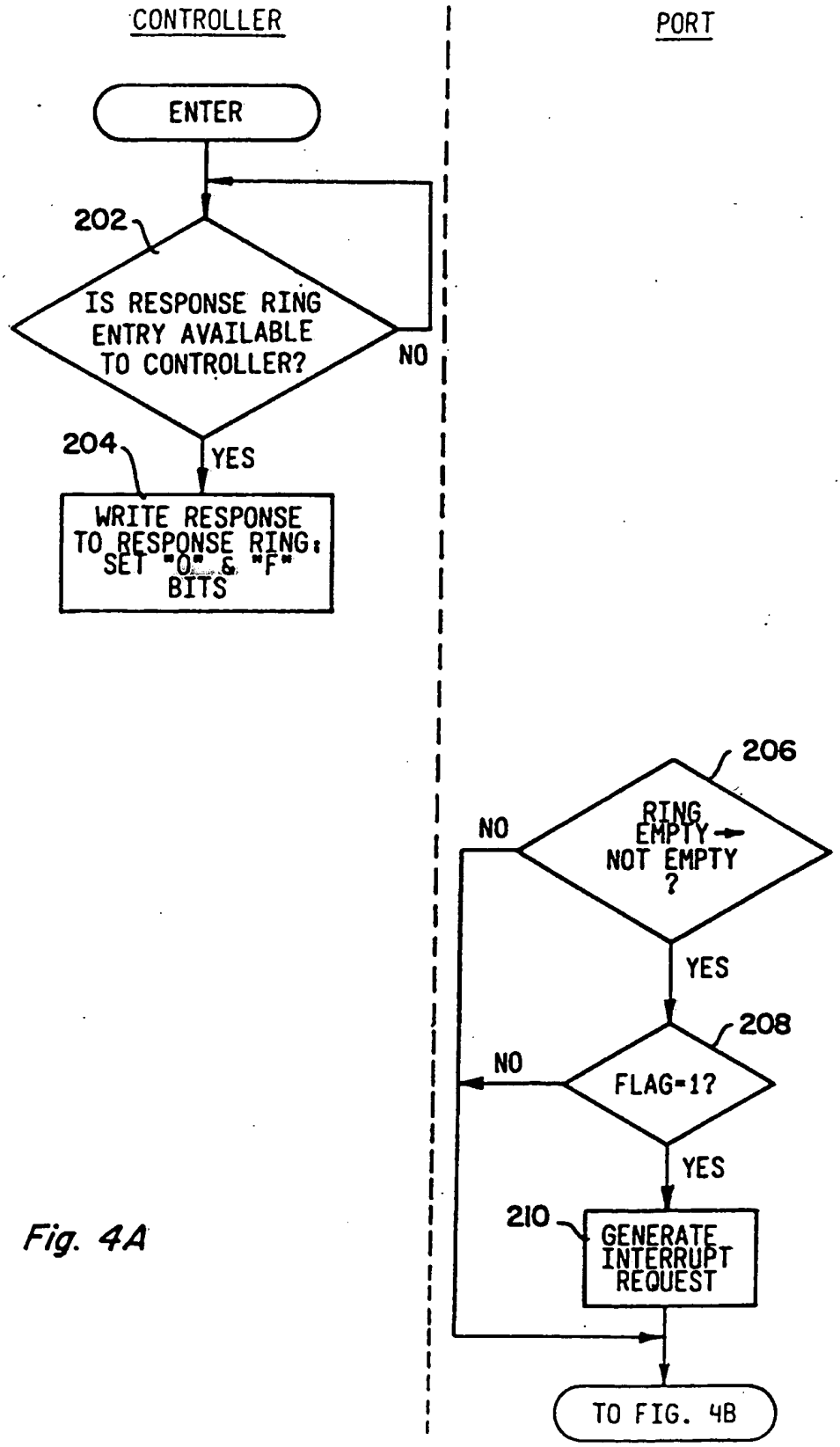
CONTROLLER
PORT
ENTER
202
IS RESPONSE RING ENTRY AVAILABLE TO CONTROLLER?
NO
YES
204
WRITE RESPONSE TO RESPONSE RING; SET "O" & "F" BITS
206
RING EMPTY → NOT EMPTY ?
NO
YES
208
FLAG=1?
NO
YES
210
GENERATE INTERRUPT REQUEST
TO FIG. 4B

Fig. 4A

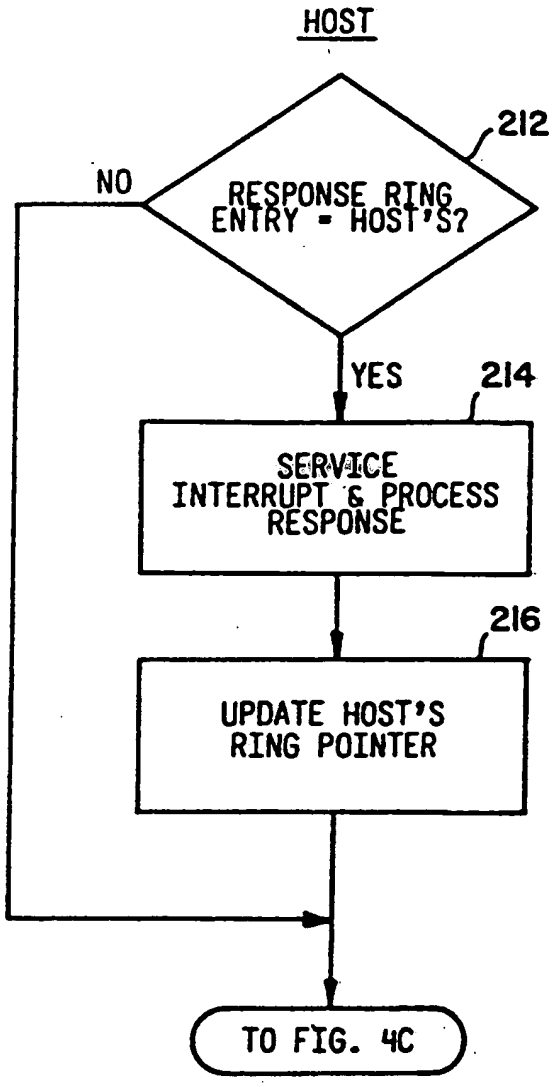
HOST
212
NO
RESPONSE RING ENTRY = HOST'S?
YES
214
SERVICE INTERRUPT & PROCESS RESPONSE
216
UPDATE HOST'S RING POINTER
TO FIG. 4C

Fig. 4B

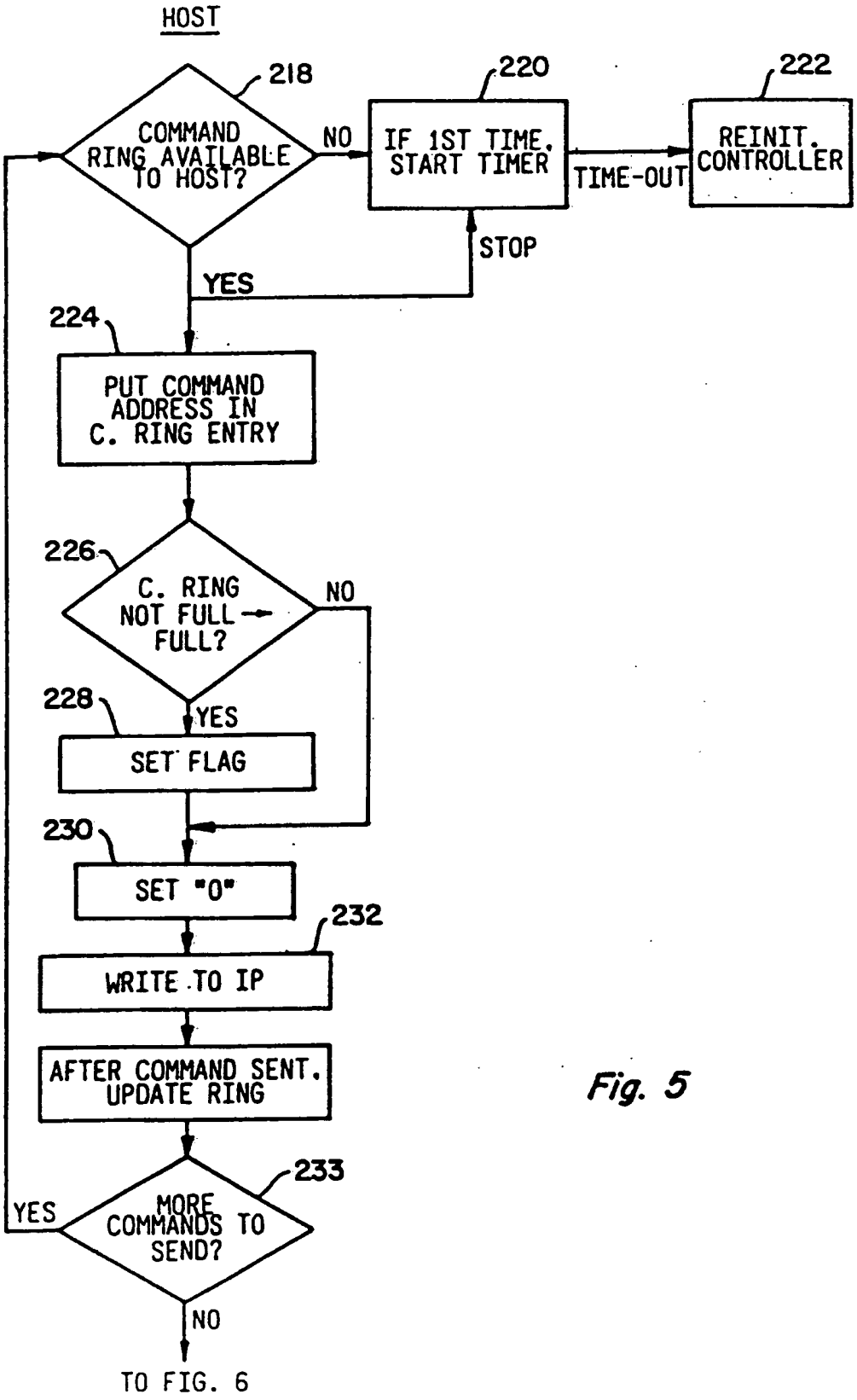
HOST
218
COMMAND RING AVAILABLE TO HOST?
NO
220
IF 1ST TIME, START TIMER
TIME-OUT
222
REINIT. CONTROLLER
STOP
YES
224
PUT COMMAND ADDRESS IN C. RING ENTRY
226
C. RING NOT FULL → FULL?
NO
228
YES
SET FLAG
230
SET "0"
232
WRITE TO IP
AFTER COMMAND SENT, UPDATE RING
233
YES
MORE COMMANDS TO SEND?
NO
TO FIG. 6

Fig. 5

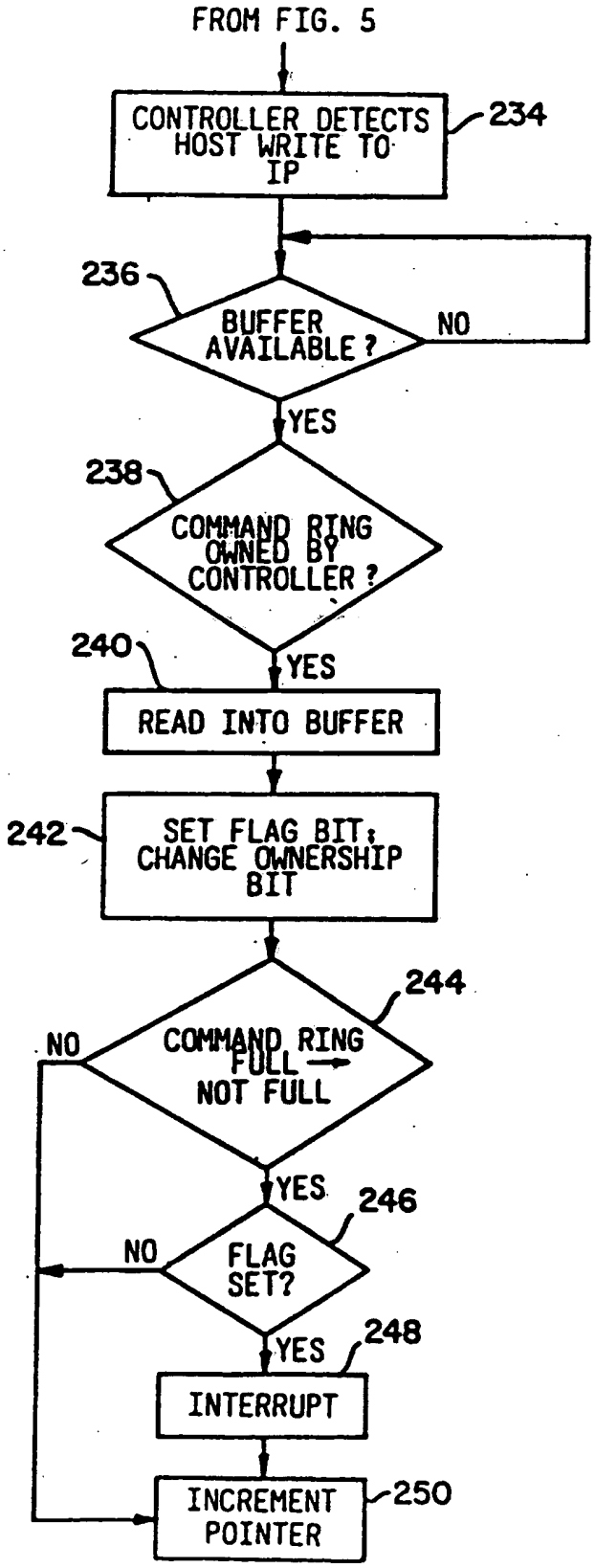
FROM FIG. 5
CONTROLLER DETECTS HOST WRITE TO IP
234
236
BUFFER AVAILABLE ?
NO
YES
238
COMMAND RING OWNED BY CONTROLLER ?
240
YES
READ INTO BUFFER
242
SET FLAG BIT; CHANGE OWNERSHIP BIT
244
NO
COMMAND RING FULL → NOT FULL
YES
246
NO
FLAG SET?
YES
248
INTERRUPT
INCREMENT POINTER
250

Fig. 6

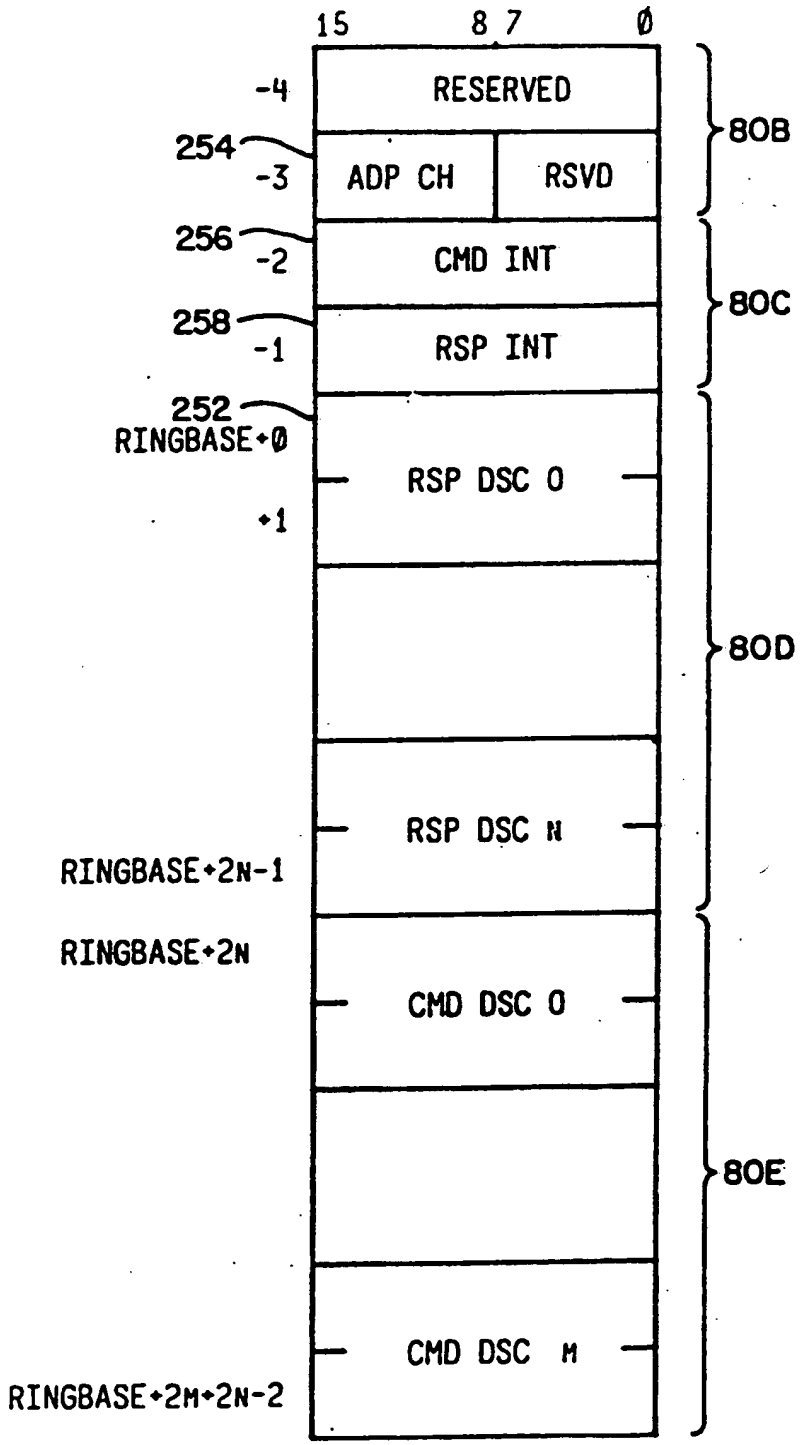
15
8 7
0
-4
RESERVED
254
-3
ADP CH
RSVD
256
-2
CMD INT
258
-1
RSP INT
252
RINGBASE+0
RSP DSC 0
+1
RSP DSC N
RINGBASE+2N-1
RINGBASE+2N
CMD DSC 0
CMD DSC M
RINGBASE+2M+2N-2
80B
80C
80D
80E

Fig. 7

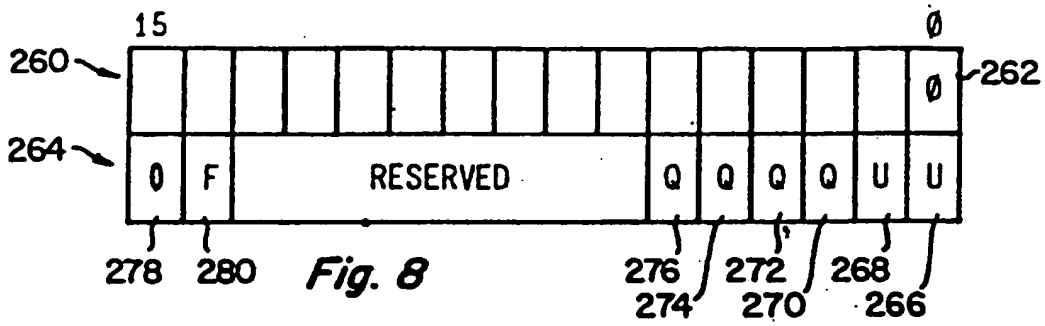

Fig. 8

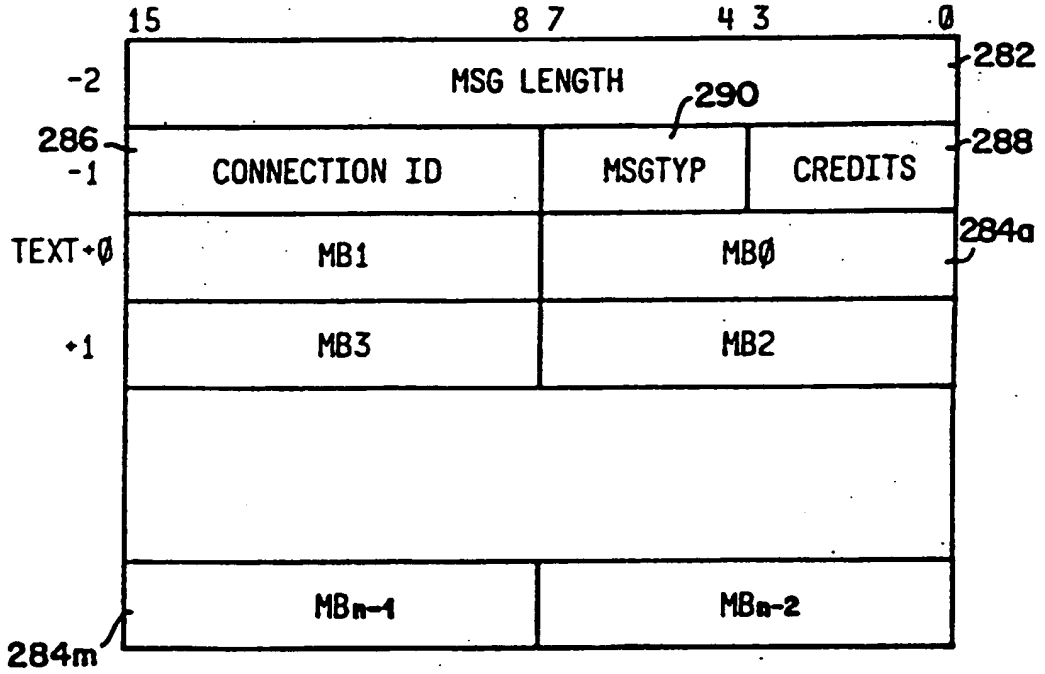

Fig. 9

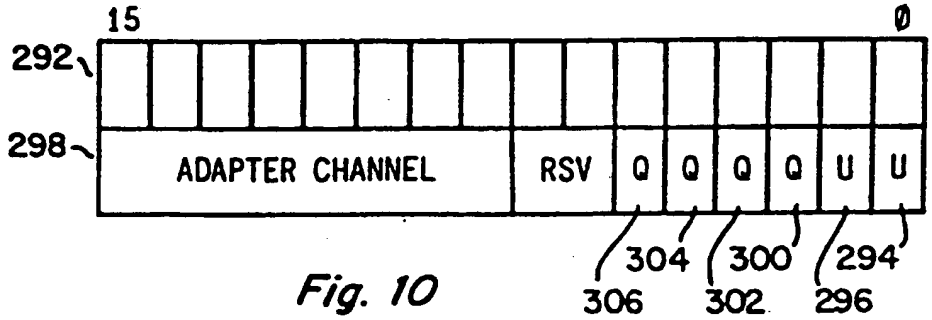

Fig. 10

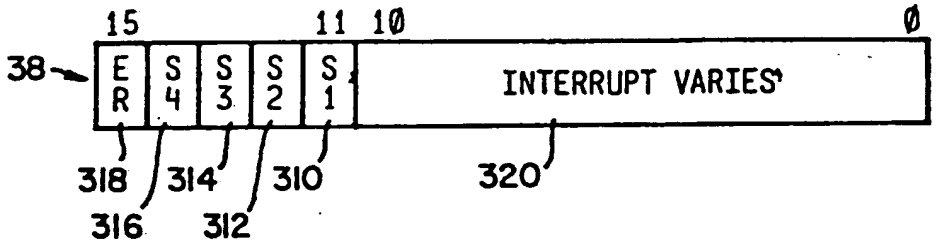
15
11
10
0
38
ER
S4
S3
S2
S1
INTERRUPT VARIES
318
314
310
320
316
312

Fig. 11

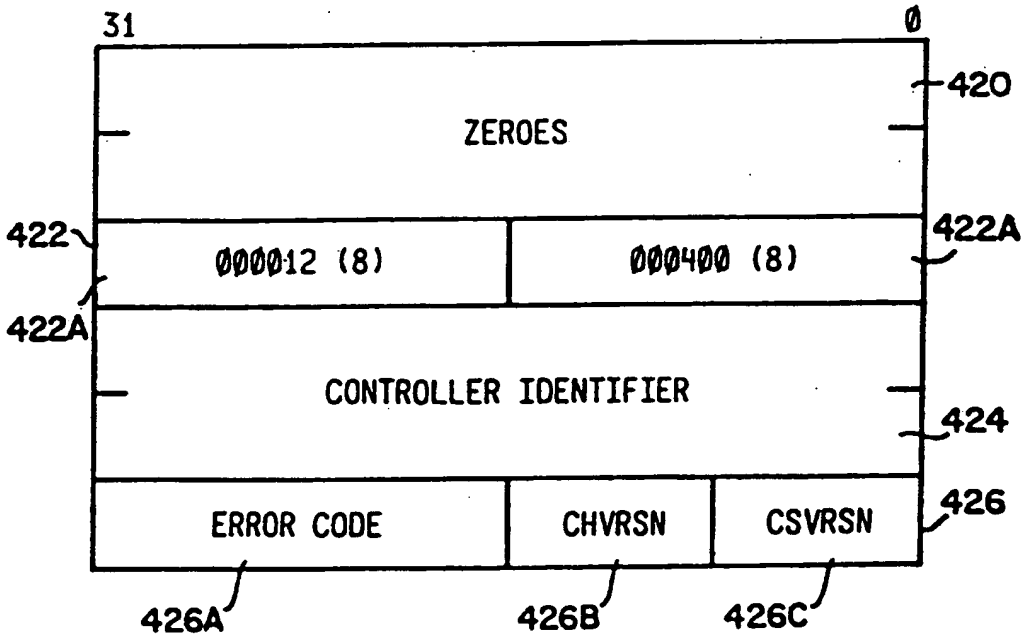
31
0
420
ZEROES
422
422A
000012 (8)
000400 (8)
422A
CONTROLLER IDENTIFIER
424
ERROR CODE
CHVRSN
CSVRSN
426
426A
426B
426C

Fig. 13

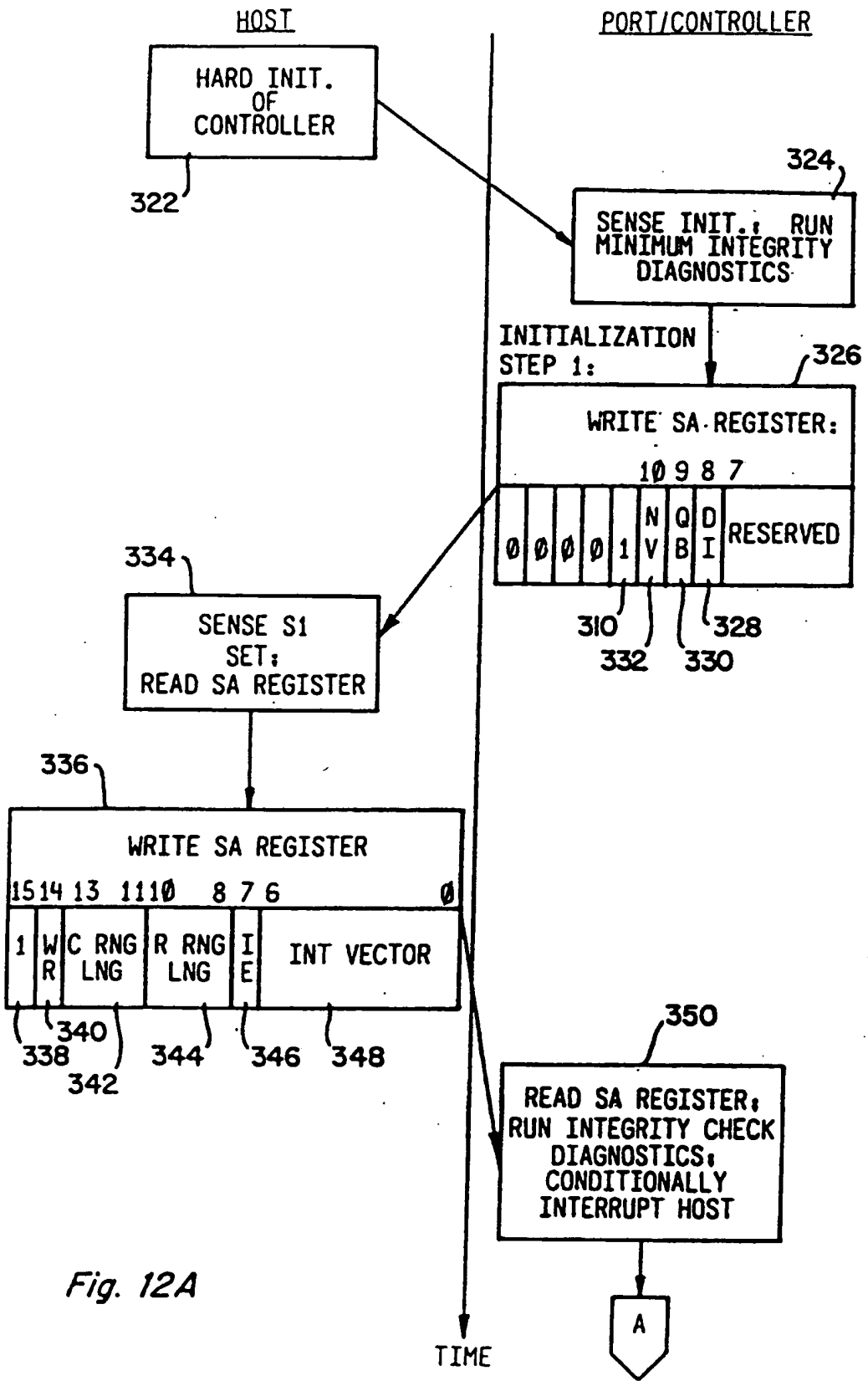
HOST
PORT/CONTROLLER
HARD INIT. OF CONTROLLER
322
324
SENSE INIT.; RUN MINIMUM INTEGRITY DIAGNOSTICS
INITIALIZATION STEP 1:
326
WRITE SA REGISTER:
10 9 8 7
0 0 0 0 1 NV QB DI RESERVED
310
332
330
328
334
SENSE S1 SET; READ SA REGISTER
336
WRITE SA REGISTER
15 14 13 11 10 8 7 6 0
1 WR C RNG LNG R RNG LNG IE INT VECTOR
338
340
342
344
346
348
350
READ SA REGISTER; RUN INTEGRITY CHECK DIAGNOSTICS; CONDITIONALLY INTERRUPT HOST
A
TIME

Fig. 12A

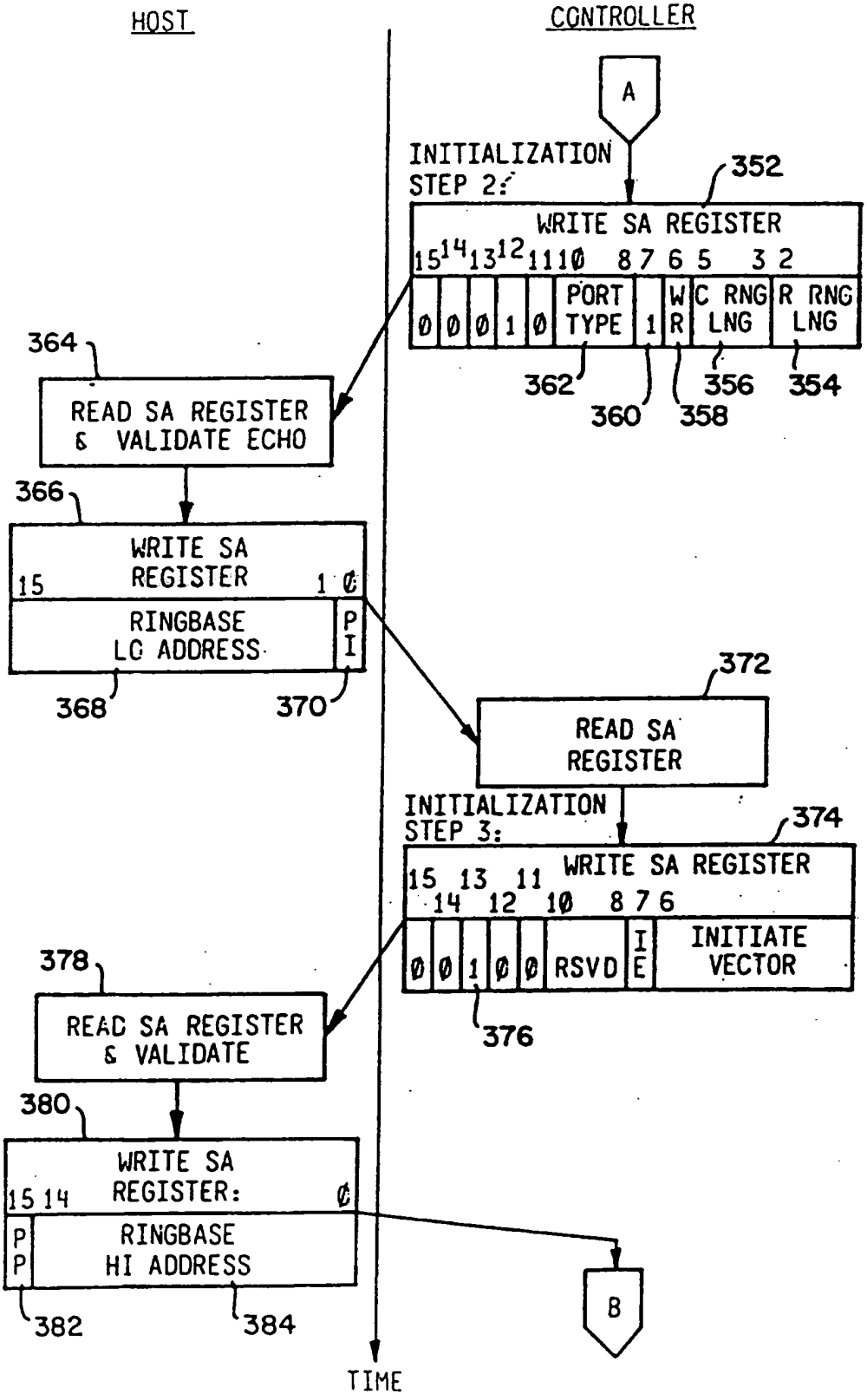
HOST
CONTROLLER
A
INITIALIZATION STEP 2:
352
WRITE SA REGISTER
15 14 13 12 11 10 8 7 6 5 3 2
0 0 0 1 0 PORT TYPE 1 WR C RNG LNG R RNG LNG
362
360
358
356
354
364
READ SA REGISTER & VALIDATE ECHO
366
WRITE SA REGISTER
15 1 0
RINGBASE LC ADDRESS
PI
368
370
372
READ SA REGISTER
INITIALIZATION STEP 3:
374
WRITE SA REGISTER
15 14 13 12 11 10 8 7 6
0 0 1 0 0 RSVD IE INITIATE VECTOR
376
378
READ SA REGISTER & VALIDATE
380
WRITE SA REGISTER:
15 14 0
PP
RINGBASE HI ADDRESS
382
384
B
TIME

Fig. 12B

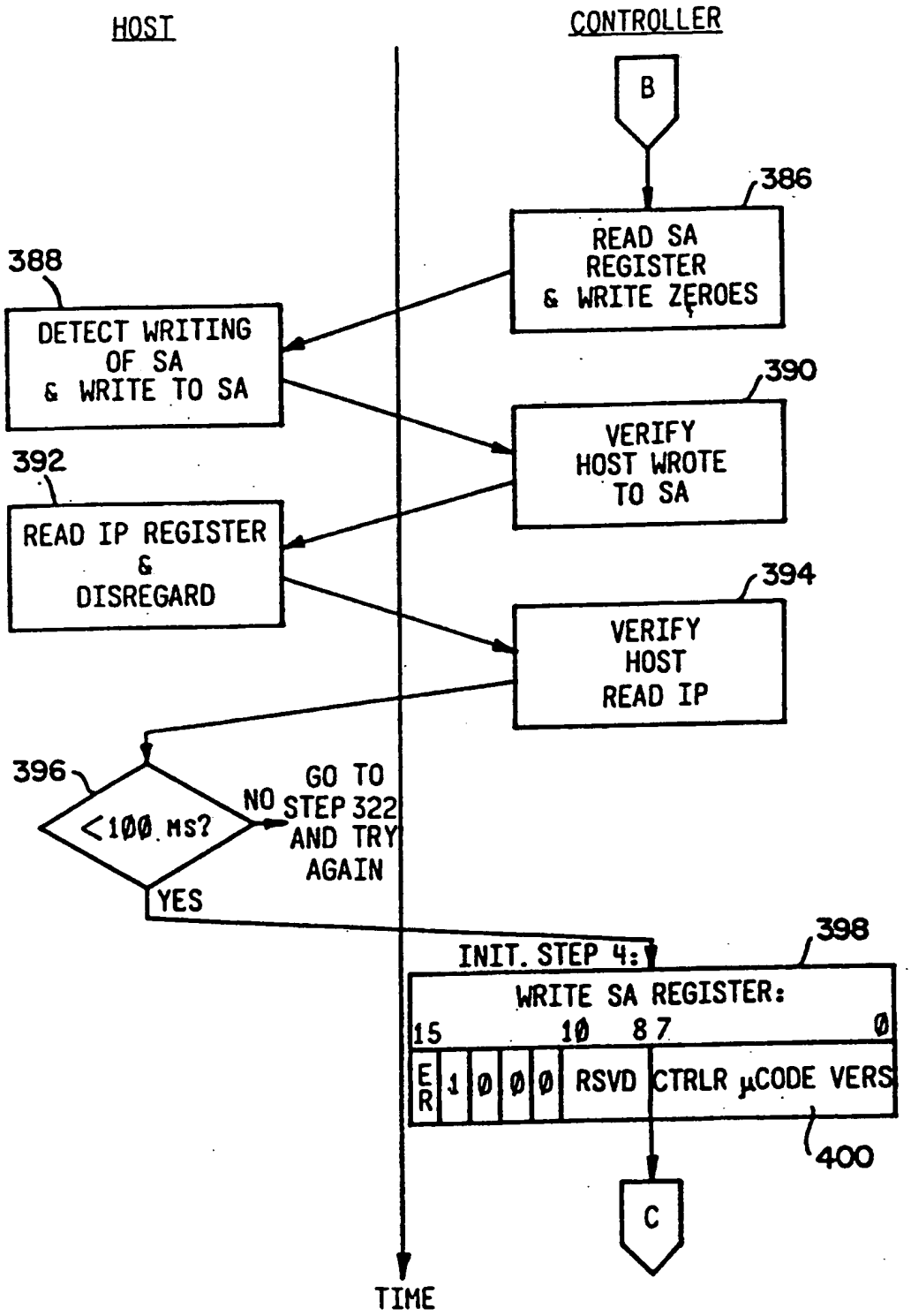
HOST
CONTROLLER
B
386
READ SA REGISTER & WRITE ZEROES
388
DETECT WRITING OF SA & WRITE TO SA
390
VERIFY HOST WROTE TO SA
392
READ IP REGISTER & DISREGARD
394
VERIFY HOST READ IP
396
<100 ms?
NO
GO TO STEP 322 AND TRY AGAIN
YES
398
INIT. STEP 4:
WRITE SA REGISTER:
15
10
8 7
0
ER
1
0
0
0
RSVD
CTRLR µCODE VERS
400
C
TIME

Fig. 12C

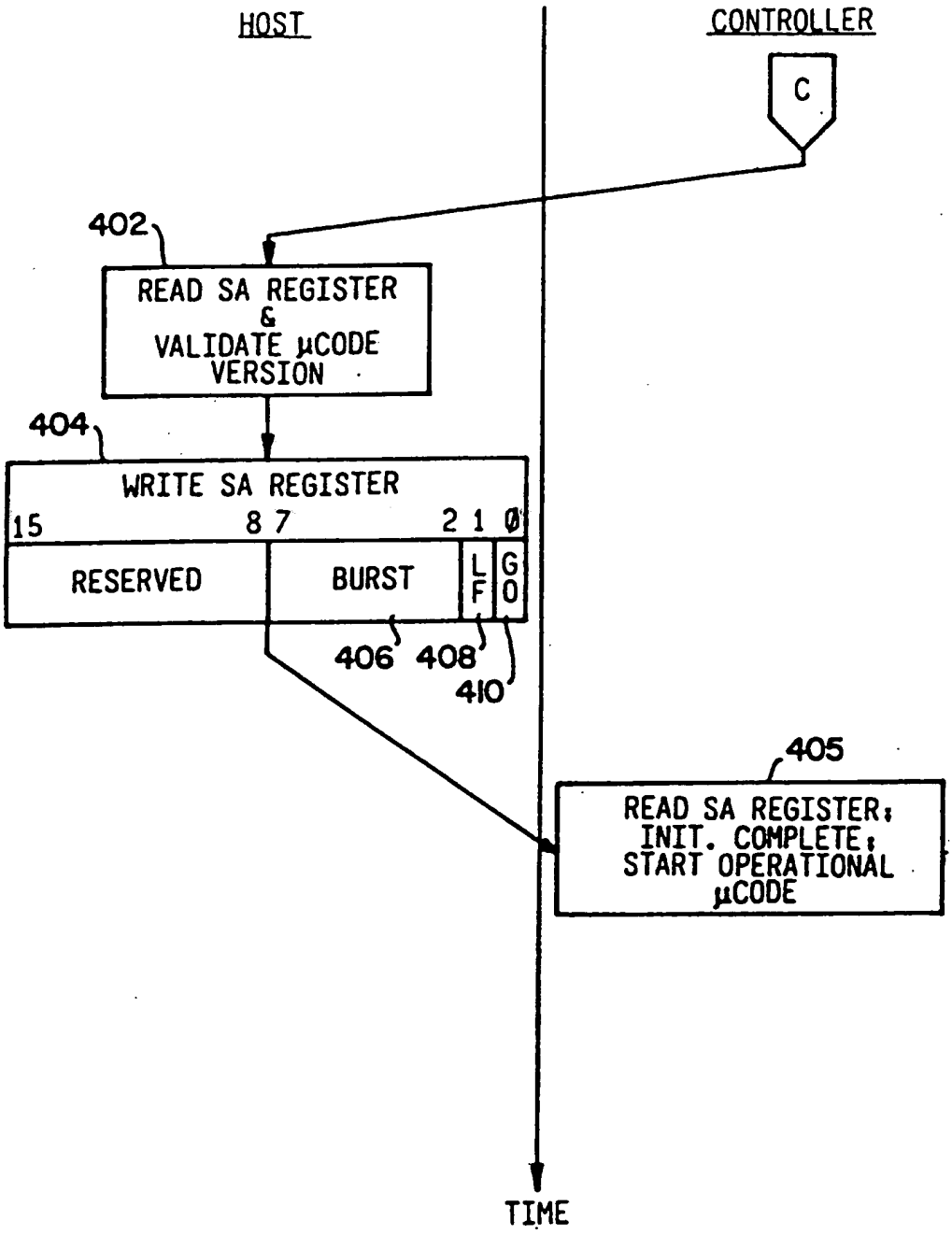
HOST
CONTROLLER
C
402
READ SA REGISTER
&
VALIDATE µCODE
VERSION
404
WRITE SA REGISTER
15
8 7
2 1 Ø
RESERVED
BURST
LF
GO
406
408
410
405
READ SA REGISTER;
INIT. COMPLETE;
START OPERATIONAL
µCODE
TIME

Fig. 12D